



Plug & Trust MW Documentation

Release v02.14.00

NXP

Apr 03, 2020

CONTENTS

1	NXP Plug & Trust Middleware	1
1.1	Organization of Documentation	2
1.2	Folder Structure	3
1.3	List of Platform Prerequisites	3
2	Changes	5
2.1	Pending Refactoring items	5
2.2	Known limitations	5
2.3	Release v02.14.00	5
2.3.1	File/Folder relocation	5
2.3.2	Build system changes	6
2.3.3	New platform support	6
2.3.4	APIs & enum/types Changes	6
2.3.5	Functional Changes	7
2.3.6	New feature support	7
2.3.7	SSSCLI / PyCLI Changes	7
2.3.8	Communication Layer Changes	7
2.3.9	APIs & enum/types Changes	8
2.3.10	Examples / DEMO updates	8
2.3.11	Documentation Changes	8
2.3.12	Other Miscellaneous Changes	9
2.4	Release v02.12.05	9
2.5	Release v02.12.04	9
2.6	Release v02.12.03	9
2.7	Release v02.12.02	9
2.7.1	AKM Specific changes	9
2.7.2	Example Updates	9
2.7.3	Other Miscellaneous Changes	9
2.8	Release v02.12.01	10
2.8.1	AKM Specific changes	10
2.8.2	New example / example updates	10
2.8.3	Other Miscellaneous Changes	10
2.9	Release v02.12.00	10
2.9.1	New feature support	10
2.9.2	SSSCLI / PyCLI Changes	10
2.9.3	Documentation Changes	11
2.10	Release v02.11.03	11
2.10.1	File/Folder relocation	11
2.10.2	New feature support	11
2.10.3	Scripts and Build changes	11

2.10.4	Documentation Changes	11
2.10.5	Other Miscellaneous Changes	12
2.11	Internal Release v02.11.01	12
2.11.1	APIs & enum/types Changes	12
2.11.2	Functional Changes	12
2.11.3	New feature support	13
2.11.4	Scripts and Build changes	13
2.11.5	Documentation Changes	13
2.11.6	Communication Layer Changes	13
2.11.7	User Interface Changes	13
2.11.8	Other Miscellaneous Changes	14
2.12	Release v02.11.00	14
2.12.1	File/Folder relocation	14
2.12.2	APIs & enum/types Changes	16
2.12.3	Functional Changes	16
2.12.4	New platform support	16
2.12.5	Scripts and Build changes	16
2.12.6	SSSCLI / PyCLI Changes	16
2.12.7	Documentation Changes	17
2.12.8	Communication Layer Changes	17
2.12.9	Other Miscellaneous Changes	17
2.13	Release v02.10.00	18
2.13.1	APIs & enum/types Changes	18
2.13.2	New platform support	19
2.13.3	Scripts and Build changes	19
2.13.4	SSSCLI / PyCLI Changes	19
2.13.5	Other Miscellaneous Changes	19
2.14	Release v02.09.00	19
2.14.1	Middleware	19
2.14.2	CLI Tool	19
2.15	Release v02.07.00	20
2.15.1	Middleware	20
2.15.2	SMCom	20
2.15.3	SSS APIs	20
2.15.4	CLI Tool	20
2.15.5	Development Platform	21
2.16	Release v02.06.00	21
2.16.1	Demos	21
2.16.2	Plug & Trust Middleware	21
2.16.3	Documentation	21
2.16.4	Development Platform	21
2.17	Release v02.05.00	21
2.18	Release v02.04.00	22
2.19	Release 02.03.00	22
3	Plug & Trust MW Stack	23
3.1	Features	23
3.2	Plug & Trust MW : Block Diagram	24
3.3	SSS APIs	24
3.3.1	SSS: Introduction	24
3.3.2	Session	25
3.3.3	Key Store	30
3.3.4	Key Object	33
3.3.5	Asymmetric	40

3.3.6	Policies	49
3.3.7	Example Boot-Up	56
3.3.8	SSS api key format (asymmetric keys)	60
3.4	Parameter Check & Conventions	61
3.4.1	Parameter Convention	61
3.4.2	Helper Macros	62
3.4.3	Apis	62
3.5	I2CM / Secure Sensor	65
3.5.1	Normal Read/Write	65
3.5.2	Attested Read	66
3.5.3	Transaction	66
3.5.4	Read with Attestation	67
3.5.5	I2C Master APIs	67
3.6	Logging	72
3.6.1	Logging level	72
3.6.2	Adding log messages into the source code	72
3.6.3	Logging APIs	72
3.6.4	Logging Header Files	75
3.6.5	Changing logging level	75
3.7	Using Platform SCP Keys from File System	77
3.7.1	How to Run examples with Platform SCP03 keys	78
3.8	Auth Objects	78
3.9	Auth Objects : UserID	78
3.9.1	User ID - Provisioning / Injection	78
3.9.2	User ID - Use for connection / authentication	79
3.9.3	User ID - Applet Spec Notes	80
3.10	Auth Objects : AESKey	80
3.11	Auth Objects : ECKey	80
3.11.1	ECKey - Keys Used	80
3.11.2	ECKey - Use for connection / authentication	81
3.12	Key Id Range and Purpose	83
3.13	Trust provisioned KeyIDs	83
4	Building / Compiling	85
4.1	Windows Build	85
4.1.1	Prerequisite	85
4.1.2	Create Build files	85
4.1.3	SSS Examples	85
4.2	Import MCUXpresso projects from SDK	86
4.2.1	Prerequisite	86
4.2.2	Importing an example	86
4.2.3	Running / Debugging example	91
4.2.4	Logging on console	95
4.3	Freedom K64F Build (CMake - Advanced)	95
4.3.1	Prerequisite	96
4.3.2	Importing the Project	96
4.3.3	Running / Debugging example	101
4.3.4	Logging on console	105
4.4	i.MX Linux Build	105
4.4.1	Prerequisite	105
4.4.2	Build Instructions	106
4.4.3	SSS Examples	107
4.5	Raspberry Pi Build	107
4.5.1	Prerequisite	107

4.5.2	Build Instructions	107
4.5.3	SSS Examples	108
4.5.4	Enable Pin configuration for SE05X	109
4.6	CMake	109
4.6.1	Reference Commands	109
4.6.2	CMake - Cross compiling	110
4.7	CMake Options	110
4.7.1	Applet	110
4.7.2	SE05X_Ver	110
4.7.3	Host	110
4.7.4	SMCOM	111
4.7.5	HostCrypto	111
4.7.6	RTOS	112
4.7.7	mbedTLS_ALT	112
4.7.8	SCP	112
4.7.9	SE05X_Auth	112
4.7.10	A71CH_AUTH	113
4.7.11	Log	113
4.7.12	CMAKE_BUILD_TYPE	113
4.7.13	Feature Control	113
4.7.14	Deprecated Defines	114
4.7.15	NXP Internal Options	116
4.7.16	Other Variables	116
5	Demo and Examples	119
5.1	DEMO List	119
5.1.1	Platforms List	119
5.1.2	SSS APIs Examples	120
5.1.3	Cloud connectivity Examples	121
5.1.4	OpenSSL Engine Examples	121
5.1.5	mbedTLS Examples	122
5.1.6	OPC UA Examples	122
5.1.7	SE05X Specific Examples	123
5.1.8	NFC (DESFire) Examples	124
5.1.9	Examples that use OpenSSL	124
5.1.10	Ease of Use Configuration Examples	124
5.2	SSS API Examples	125
5.2.1	ECC Example	125
5.2.2	RSA Example	126
5.2.3	Symmetric AES Example	127
5.2.4	HKDF Example	128
5.2.5	Message Digest Example	129
5.2.6	HMAC Example	130
5.2.7	ECDH Example	131
5.3	AWS Demo for KSDK	131
5.3.1	Prerequisites	131
5.3.2	Using WiFi with LPC55S	132
5.3.3	Creating a device on AWS account	132
5.3.4	Creating and updating device keys and certificates to SE	132
5.3.5	Running the Demo	132
5.4	AWS Demo for iMX Linux / RaspberryPi	133
5.4.1	Prerequisites	133
5.4.2	Preparing the credentials and Provisioning the secure element	133
5.4.3	Run the example	134

5.5	GCP Demo for KSDK	136
5.5.1	Prerequisites	136
5.5.2	Using WiFi with LPC55S	136
5.5.3	Creating and updating device keys and certificates to SE	136
5.5.4	Preparing the Cloud	136
5.5.5	Running the Demo	137
5.5.6	Appendix	139
5.6	GCP Demo for iMX Linux / Raspberry Pi	139
5.6.1	Prerequisites	139
5.6.2	Preparing the credentials and Provision the SE	140
5.6.3	Building the application	140
5.6.4	Appendix	141
5.7	IBM Watson Demo for KSDK	141
5.7.1	Prerequisites	141
5.7.2	Using WiFi with LPC55S	141
5.7.3	Setting up IBM Watson IoT Platform	142
5.7.4	Creating and updating device keys and certificates to SE	143
5.7.5	Running the Demo	143
5.7.6	Appendix	144
5.8	IBM Watson Demo for iMX Linux / Raspberry Pi	144
5.8.1	Prerequisites	145
5.8.2	Preparing the credentials	145
5.8.3	Build the OpenSSL engine [Optional]	145
5.8.4	Running the Demo on iMX/Raspberry Pi	146
5.8.5	Appendix	147
5.9	Azure Demo for KSDK	147
5.9.1	Prerequisites	147
5.9.2	Using WiFi with LPC55S	147
5.9.3	Creating a device on azure IoT Hub portal	147
5.9.4	Creating and updating device keys and certificates to SE	148
5.9.5	Uploading root certificates to IoT Hub	148
5.9.6	Running the Demo	148
5.9.7	Appendix	149
5.10	Azure Demo for iMX Linux / Raspberry Pi	149
5.10.1	Prerequisites	150
5.10.2	Preparing the credentials and Provisioning the secure element	150
5.10.3	Registering Device	150
5.10.4	Create device enrollment in azure IoT Hub portal	150
5.10.5	Uploading root certificates to IoT Hub	151
5.10.6	Build the OpenSSL engine [Optional]	152
5.10.7	Run the example	152
5.11	Greengrass Demo for Linux	153
5.11.1	Prerequisites	153
5.11.2	Preparing the Greengrass group	154
5.11.3	Provisioning SE050 and Building PKCS#11 library	154
5.11.4	Updating Greengrass configuration	155
5.11.5	Running Greengrass Core	156
5.11.6	Connecting Devices to Greengrass Core	156
5.11.7	Troubleshooting	156
5.12	OpenSSL Engine: TLS Client example for iMX/Rpi3	157
5.12.1	Summary	157
5.12.2	Credential preparation (execute once) [Optional]	157
5.12.3	Secure Element preparation (client side)	158
5.12.4	Server side preparation	159

5.12.5	Start up the server	159
5.12.6	Establish a TLS link from the client to the server	159
5.12.7	TLS client example using A71CH	160
5.13	OPC UA (Open62541) Demo	161
5.13.1	Supported Platforms	161
5.13.2	Introduction	161
5.13.3	Build Open62541 server and client examples	162
5.13.4	Test Open62541 server and client examples	163
5.13.5	Known Limitations	164
5.14	SE05X Minimal example	164
5.14.1	Prerequisites	164
5.14.2	Building the Demo	164
5.14.3	Running the Example	165
5.14.4	Console output	165
5.15	SE05X Get Info example	165
5.15.1	Prerequisites	165
5.15.2	Building the Demo	165
5.15.3	Running the Example	165
5.15.4	Console output	166
5.16	APDU Player Demo	167
5.16.1	How to use	167
5.17	Using policies for secure objects	167
5.17.1	Sign Policy	168
5.17.2	Using PCR Object	169
5.17.3	Console output	170
5.18	Get Certificate from the SE	170
5.18.1	Building the example	171
5.18.2	How to use	171
5.19	SE05X Rotate PlatformSCP Keys Demo	171
5.19.1	Prerequisites	172
5.19.2	Configuring the Demo	172
5.19.3	Building the Demo	173
5.19.4	Running the Example	173
5.19.5	Console output	173
5.20	I2C Master Example	173
5.20.1	Prerequisites	173
5.20.2	About the Example	176
5.20.3	Running the Demo	177
5.21	SE05X WiFi KDF Example	177
5.21.1	Prerequisites	177
5.21.2	Building the Demo	178
5.21.3	Running the Example	178
5.21.4	Console output	178
5.22	SE05X Import Transient objects	178
5.22.1	Visual Studio - Debug settings	179
5.22.2	Console output	179
5.23	SE05X Export Transient objects	179
5.23.1	Visual Studio - Debug settings	180
5.23.2	Console output	180
5.24	Import External Object Prepare	180
5.24.1	Building	182
5.24.2	How to use	182
5.25	Import External Object Create	183
5.25.1	Pre-requisites	183

5.25.2	Building	183
5.25.3	How to use	183
5.26	SE05X Mandate SCP example	183
5.26.1	Prerequisites	184
5.26.2	Building the Demo	184
5.26.3	Running the Example	184
5.27	Read object with Attestation	184
5.27.1	Building	184
5.27.2	Running	185
5.27.3	Reading large binary objects with attestation	185
5.28	SE05X Transport Lock example	188
5.29	SE05X Transport UnLock example	188
5.30	SE05X Timestamp	189
5.30.1	Building	189
5.30.2	Running	189
5.31	MIFARE DESFire EV2 : Prepare SE050	189
5.31.1	Prerequisites	189
5.31.2	About the Example	189
5.31.3	The two AES key storage for NFC application n SE050	190
5.31.4	Running the Demo	190
5.32	MIFARE DESFire EV2 : Prepare MFDFEV2	190
5.32.1	Prerequisites	190
5.32.2	About the Example	191
5.32.3	Creation of application	191
5.32.4	AES Keys provisioned in card	191
5.32.5	Running the Demo	191
5.33	MIFARE DESFire EV2 : Authentication	193
5.33.1	Prerequisites	193
5.33.2	About the Example	193
5.33.3	Running the Demo	193
5.34	MIFARE DESFire EV2 : Change Key	196
5.34.1	Prerequisites	196
5.34.2	About the Example	196
5.34.3	Running the Demo	197
5.35	MIFARE DESFire EV2 : Diversified Change Key	206
5.35.1	Prerequisites	206
5.35.2	About the Example	206
5.35.3	Running the Demo	207
5.36	Tool to create Reference key file	217
5.36.1	Building the example	218
5.36.2	How to use	218
5.37	Building a self-signed certificate	219
5.37.1	How to use	219
5.38	Write APDU to buffer	219
5.38.1	Building	220
5.38.2	Running	220
5.39	Ease of Use configuration - IBM Watson	221
5.39.1	Configuring Device type and Device name	221
5.39.2	Uploading certificate chain	221
5.39.3	Running the Demo	221
5.39.4	Update cloud example	222
5.39.5	Build and run the demo.	222
5.40	Ease of Use configuration - Google Cloud Platform	223
5.40.1	Pre-requisites	223

5.40.2	Creating Registry and Devices	223
5.40.3	Running the Demo	223
5.40.4	Update cloud example	224
5.40.5	Build and run the demo.	224
5.41	Ease of Use configuration - Azure IoT Hub	225
5.41.1	Creating Device on Azure DPS	225
5.41.2	Registering Device to IoT Hub	225
5.41.3	Running Azure Demo	226
5.41.4	Update cloud example	226
5.41.5	Build and run the demo.	227
5.42	Ease of Use configuration - AWS IoT Console	227
5.42.1	Pre-requisites	227
5.42.2	Extracting Device Certificate	227
5.42.3	Registering Device Certificate	228
5.42.4	Running on Linux	228
5.42.5	Update Cloud Example	229
5.42.6	Build and run the demo.	229
6	Plugins / Add-ins	231
6.1	Introduction on OpenSSL engine	231
6.1.1	General	231
6.1.2	Keys	232
6.1.3	Building the OpenSSL engine	234
6.1.4	Sample scripts to demo OpenSSL Engine	234
6.2	Introduction on mbedTLS ALT Implementation	239
6.2.1	Using mbedTLS ALT	239
6.2.2	Testing	240
6.2.3	mbedTLS ALT APIs	241
6.3	Platform Security Architecture	242
6.3.1	PSA SE Driver Interface	242
6.3.2	PSA Concepts	243
6.3.3	Building PSA for TrustZone	244
6.4	Android Key master	244
6.5	Introduction on Open62541 (OPC UA stack)	244
6.5.1	Integrating SE050 in Open62541	244
6.6	WiFi EAP Demo with Raspberry Pi3	245
6.6.1	Prerequisites	245
6.6.2	Introduction	245
6.6.3	Setting up Access point	245
6.6.4	Setting up freeradius Server on Ubuntu	246
6.6.5	Setting up Raspberry Pi3	248
6.7	PKCS#11 Standalone Library	249
6.7.1	Building on Linux/Raspberry Pi3	249
6.7.2	PKCS#11 specifications	250
6.7.3	Using with pkcs11-tool	251
7	CLI Tool	253
7.1	Introduction	253
7.2	Block Diagram	254
7.3	Steps needed before running <code>sscli</code> tool	254
7.3.1	Once per installation	254
7.3.2	On change of interface	256
7.4	Running the <code>sscli</code> tool - Windows	256
7.5	CLI Provisioning	256

7.5.1	Generating keys and certificates	256
7.5.2	Provisioning for the demo	257
7.5.3	Steps to provision your device for demo on Windows	257
7.5.4	Steps to provision your device for demo on iMX or Raspberry Pi	258
7.6	Usage Examples	259
7.6.1	SE050: VCOM Interface	259
7.6.2	SE050: PCSC interface	261
7.6.3	SE050: JRCPV2 interface	261
7.6.4	A71CH: VCOM Interface	264
7.6.5	A71CH: SCI2C interface	264
7.6.6	MBEDTLS	265
7.7	List of <code>sscli</code> commands	265
7.7.1	SSSCLI Commands	266
7.7.2	Set Commands	271
7.7.3	Get Commands	273
7.7.4	Generate Commands	274
7.7.5	Refpem Commands	275
7.7.6	Se05x Commands	276
7.7.7	A71CH Commands	277
7.8	CLI Data formats	277
7.8.1	DER	277
7.8.2	PEM	278
7.8.3	HEX	278
7.8.4	REFERENCE KEY	278
7.9	Upload keys and certificates to SE05X using Pycli tool	279
8	A71CH	281
8.1	A71CH and SSS API	281
8.1.1	Introduction	281
8.1.2	A71CH API to SSS API mapping	281
8.1.3	Mixing SSS API and A71CH API	283
8.1.4	SSS Object Identifier to A71CH Internal storage mapping	285
8.2	Miscellaneous	285
8.2.1	Demos and examples supported on A71CH	285
8.2.2	OpenSSL Engine	286
8.2.3	A71CH and SCP03	287
8.2.4	A71CH on Raspberry Pi	287
8.3	A71CH Legacy API	288
8.3.1	Introduction	288
8.3.2	A71CH API	288
8.3.3	SW structure	289
8.3.4	API details	290
8.4	A71CH Legacy HLSE (Generic) API	314
8.4.1	HLSE API	314
8.4.2	Logical objects	315
8.4.3	API details	318
8.5	A71CH Legacy Configure Tool	329
8.5.1	Introduction	329
8.5.2	Usage modes	331
8.5.3	Tool deployment	331
8.5.4	Command reference	334
8.5.5	Not connected mode	345
9	Appendix	347

9.1	Glossary	347
9.2	APDU Commands over VCOM	347
9.2.1	COM port parameters	347
9.2.2	VCOM Format	347
9.2.3	Example Commands	348
9.3	Visual Studio 2019 Setup	350
9.3.1	Prerequisites	350
9.3.2	Installing the components	350
9.4	Setting up MCUXpresso IDE	352
9.4.1	To Download and install MCUXpresso IDE:	352
9.4.2	To Install board specific SDK in MCUXpresso	353
9.5	Development Platforms	353
9.5.1	Setup i.MX 8MQuad - MCIMX8M-EVK	353
9.5.2	Setup i.MX6UL - MCIMX6UL-EVK	359
9.5.3	Freedom K64F	362
9.5.4	Connecting SE050 Arduino shield to imxRT1050	362
9.5.5	Connecting RC663 to imxRT1050	364
9.5.6	Connecting SE050 Arduino shield to LPC55S	365
9.5.7	Connecting WiFi shield Silex-2401 to LPC55S	366
9.5.8	Connecting CLEV6630B to FRDM-K64F or LPC55S69	367
9.5.9	Android	370
9.6	How to get SE Platform Information and UID	384
9.6.1	Using TeraTerm and pre-built binary	385
9.6.2	Using VCOM and binary	386
9.6.3	Using pySSSCLI Tool	388
9.6.4	SE Platform Information on Android platform	388
9.7	Version Information	389
9.8	Certificate Chains : ROOT	390
9.8.1	ECC	390
9.8.2	RSA	391
9.9	Certificate Chains : DEV Kit	392
9.9.1	Certificate Chain for SE050	393
9.10	JRCP_v1 Server	394
9.10.1	Introduction	394
9.10.2	Using the JRCP_v1 server	395
9.11	Using own Platform SCP03 Keys	395
9.12	Write APDU to buffer	396
9.13	Plug & Trust MW APIs	396
9.13.1	Class Hierarchy	396
9.13.2	File Hierarchy	400
9.13.3	Full API	401
10	Indices and tables	665
	Index	667

NXP PLUG & TRUST MIDDLEWARE

The NXP Plug&Trust Middleware documentation covers following Secure Elements:

- EdgeLock™ SE050 (Including variants **A**, **B** and **C**)
- A71CH (refer to the *A71CH* section at the end of this document)

Setting up SE050 development environment for:

- iMX6UL, iMX8MQ - Linux
- Freedom K64F, i.MX RT 1050, LPC55S - FreeRTOS/Without RTOS
- Hikey 960 - Android
- Raspberry-Pi 3 - Raspian Linux
- Windows PC(Visual Studio)

Documentation also covers:

- Executing Demos and Examples.
- Using the CLI Tool to configure the Secure Element for the Demos.
- Setting up the iMX and Kinetis Freedom boards to be used with the CLI Tool.
- Setting up and executing Demo Examples.

Dedicated application notes:

- To assist end users from different backgrounds, dedicated application notes are prepared and available.
- Some of the important Application Notes are as shown below:
 - MCU/RTOS: [AN12396](#)
 - MPU/Linux: [AN12397](#)
 - Windows: [AN12398](#)
- More details regarding SE050 and other detailed application notes can be found at <https://www.nxp.com/products/:SE050>

1.1 Organization of Documentation

This documentation is organized into following parts/sections.

- What has changed so far in the middleware stack be found in [Section 2 — Changes](#).
- The overall middleware stack is explained in [Section 3 — Plug & Trust MW Stack](#)
 - The common SSS APIs, a subset of the overall stack, is explained in [Section 3.3 — SSS APIs](#)
- How to build MW for various reference platform/IDEs is explained in [Section 4 — Building / Compiling](#).
 - [Section 4.1 — Windows Build](#)
 - [Section 4.2 — Import MCUXPresso projects from SDK](#). Steps for i.MX RT 1050 and LPC55S are similar to those for [Import MCUXPresso projects from SDK](#)
 - For advanced users to take power of CMake's capabilities, [Section 4.3 — Freedom K64F Build \(CMake - Advanced\)](#). Steps for i.MX RT 1050 and LPC55S are similar to those for [Freedom K64F Build \(CMake - Advanced\)](#)
 - [Section 4.4 — i.MX Linux Build](#)
 - [Section 4.5 — Raspberry Pi Build](#)
- The MW package has lots of examples, they are part of [Section 5 — Demo and Examples](#).
 - [Section 5.1 — DEMO List](#) has an itemized list of the demos.
 - [Section 5.1.1 — Platforms List](#)
 - [Section 5.1.3 — Cloud connectivity Examples](#)
 - [Section 5.1.4 — OpenSSL Engine Examples](#)
 - [Section 5.1.5 — mbedTLS Examples](#)
 - [Section 5.1.6 — OPC UA Examples](#)
 - [Section 5.1.7 — SE05X Specific Examples](#)
 - [Section 5.1.8 — NFC \(DESFire\) Examples](#)
 - [Section 5.1.10 — Ease of Use Configuration Examples](#)
- Integration with other middlewares/systems is explained in [Section 6 — Plugins / Add-ins](#).
 - [Section 6.1 — Introduction on OpenSSL engine](#)
 - [Section 6.2 — Introduction on mbedTLS ALT Implementation](#)
 - [Section 9.5.9 — Android](#)
 - [Section 6.5 — Introduction on Open62541 \(OPC UA stack\)](#)
 - [Section 6.7 — PKCS#11 Standalone Library](#)
- The CLI Tool is explained in [Section 7 — CLI Tool](#)
 - [Section 7.7 — List of ssscli commands](#)

1.2 Folder Structure

Folder / File Name	Description
README . First . txt	<i>PLEASE Read this file before using the Software Package</i>
EULA.pdf	Contains the End User License Agreement. Required to be agreed mandatorily before using the PlugAndTrustMW software
Third _ Party _ Li- cense.pdf	This file lists the Third Party license(s) in text that are part of this software package.
akm	This folder has the files corresponding to Android Key Master implementation as part of Plug&Trust MW Currently AKM solution is supported only on HiKey 960 development platform.
binaries	This folder has pre-build binaries and executables. e.g. It has firmware binaries to emulate Virtual COM port on freedom K64F, etc. binaries for platforms supported. For Ex: FRDM-K64F, i.MX RT1050.
binaries/pySSCLI	This folder contains pre-compiled SSCLI/pycli tool. This tool runs on Windows.
demos	This folder contains various Demonstration Examples for the supported platforms. See DEMO List for list of supported Demos.
demos / Certifi- cate_Chains	This folder contains the certificate chains as per OEF configuration The intermediate and RootCA certificates are required to connect to clouds using trust provisioned device certificates. See Certificate Chains : ROOT
doc	This folder contains the documentation for Plug&Trust MW in the form of html files
ext	This folder contains files related to external dependencies that are required to build the Middleware and it's demo examples. e.g. amazon-FreeRTOS, openssl, mbedTLS, JRCP, etc.
hostlib	This folder contains the <i>common</i> part of host library e.g. T=1oI2C communication protocol stack, SE050 APIs, etc.
projects	This contains MCUXpresso projects that can use CMake generated gnu-arm-gcc cross compiled build directories for easy download and debug.
pycli	This folder contains the NXP proprietary command-line tool for configuring and provisioning the SE050 (Secure Element) Please ensure that PYTHON Version 3.6 or later (32 bit) is installed. The default pre-compiled DLLs are 32bit and hence this tool needs a 32 bit Python.
scripts	This folder contains scripts for creating CMake projects
sss	This folder contains the “SSS” APIs interface to the Application Layer
tools	This folder contains the pre-built binaries and DLLs.

1.3 List of Platform Prerequisites

All the examples assume that build environment is setup and refer these sections for guidance.

Platform	Link
Windows	Windows Build
FRDM-K64F	Import MCUXpresso projects from SDK or Freedom K64F Build (CMake - Advanced)
Raspberry Pi 3	Raspberry Pi Build
IMX6UL	i.MX Linux Build

CHANGES

2.1 Pending Refactoring items

The following items would be re-factored and changed in future releases. These topics are deemed to be deprecated and to be used with vigilance.

Issue key	Summary
SIMW-1437	[Linux] Unify the Linux I2C implementation
SIMW-1416	Use fsl_sss_ftr.h to select SCP/etc. (Instead of makefiles/CMakeFiles)

2.2 Known limitations

The following known limitations exist in this package. And they would be addressed in subsequent releases.

Issue key	Summary
SIMW-1047	[pyCLI] Investigate on slow loading on ssscli
SIMW-1892	Traditional openssl key format not supported in sss_key_store_set_key for openssl

2.3 Release v02.14.00

2.3.1 File/Folder relocation

- Renamed DTLS/SSL2 Server and client executables. New names are:
 - mbedtls_ex_orig_ssl_server2
 - mbedtls_ex_sss_dtls_client
 - mbedtls_ex_orig_dtls_server
 - mbedtls_ex_sss_ssl2_client
- Renamed project greengrass to sss_pkcs11
- Renamed file greengrass.c to sss_pkcs11.c
- Renamed folders of Reader Library examples.
 - ex_prepare_MFDFEV2 => ex_Ev2Prepare_Card
 - ex_prepare_se05x => ex_Ev2Prepare_se05x

2.3.2 Build system changes

- Extensively revamped `fsl_sss_ftr.h` file for finer control of build configuration selection. This design will be extended extensively in future releases.
- On LPC55S with FreeRTOS, using native `malloc` instead of `Heap_4.c` for `MBEDTLS`
- Compile time asserts added for sizes of structures.
- `scripts/env_setup.sh`, `scripts/env_setup.sh` prints info on which tools are used from which paths.
- Changed Applet selection in CMake (See [Section 4.7.1 Applet](#)). We no longer use name `SE050_A`, `SE050_B` or `SE050_C` for builds / Applet selection. New names are `SE05X_A`, `SE05X_B` or `SE05X_C`
- CMake Option `Applet_SE05X_Ver` is no longer used. Instead, [Section 4.7.2 SE05X_Ver](#) is introduced for future use.
- See [Section 4.7.9 SE05X_Auth](#)
 - FastSCP is now called ECKey.
 - AppletSCP is now called AESKey.

2.3.3 New platform support

- i.MX8 support added (*Setup i.MX 8MQuad - MCIMX8M-EVK*)

2.3.4 APIs & enum/types Changes

- Use `Se05x_API_ReadObject_W_Attest()` instead of `sss_se05x_key_store_get_key_attst()` to read with attestation large binary files greater than 500 bytes. See example *Reading large binary objects with attestation*
- For Montgomery curves the key arguments, DH Shared secret and Signature are passed in Little Endian Convention. Refer to *SSS api key format (asymmetric keys)* for details on Endianness.
- `sss_derive_key_go()` is deprecated and is replaced by `sss_derive_key_one_go()`
- Added `sss_status_sz()` to convert SSS API Return code to string.
- Updated Enumeration from `SE05x_TransientType_t` to `SE05x_INS_t` in the following API's:
 - `Se05x_API_WriteECKey`
 - `Se05x_API_WriteRSAKey`
 - `Se05x_API_WriteSymmKey`
- Define `T1oI2C_UM1225_SE050` is no longer applicable, use `T1oI2C_UM11225_SE05X` instead.
- Added SE050 APIs `Se05x_API_CreateCounter`, `Se05x_API_SetCounterValue`, `Se05x_API_IncCounter`
- smCom Layer is refactored so that Application send down the connection handles/parameters to lower layer.

e.g. SSCLI and Demos on PC which can take command line argument can now use the I2C device over command line at run time without recompiling the middleware/example.

2.3.5 Functional Changes

- Extensive support for *A71CH*.
- Added enable pin support for SE05X on Raspberry Pi
- Modified SE policy of keymaster HAL in Android
- Updated RSA reference key format for Android Key Master. It now uses prefix A5 to import import Key ID 00000001.

2.3.6 New feature support

- Added tool se05x_setAppletFeatures to configure applet features
- Added support to use Platform SCP keys from file system
- Added support to retrieve existing certificates in pem format
- Added tool to mandate Platform SCP03
- Integrated mBED Crypto PSA interface
- Added Secure-NonSecure example based on PSA for LPC55S
- Added examples of SE05X Import Transient objects, SE05X Export Transient objects, Import External Object Prepare and Import External Object Create
- Added example to demonstrate object read with attestation
- Added example to demonstrate how timestamp is incremented in SE
- Added example to demonstrate how to create APDU buffer to import external key objects.
- Lock and unlock secure element using transport key
- Upgraded mbedTLS to version 2.16

2.3.7 SSSCLI / PyCLI Changes

- Added support for ECC ED25519 and MONTH DH 25519 curves
- Fixed sign and verify operation for ED25519.
- Added API to inject HMAC key
- Endianness of ed25519 and mont_dh_25519 keys, signature and shared secret are updated to little endian.

2.3.8 Communication Layer Changes

- VCOM Interface updates on OSX and PC Linux
- Added connection handle in smCom layer. This allows connection data to be passed from application. Tested on windows, raspberry pi and imx platform.

2.3.9 APIs & enum/types Changes

- Re-Wrote (internal) low level Tx/Rx APIs for APDU TxRx.
 - 1) DoAPDUTxRx_s_Case2
 - 2) DoAPDUTxRx_s_Case3
 - 3) DoAPDUTxRx_s_Case4
 - 4) DoAPDUTxRx_s_Case4E
- Define T1oI2C_UM1225_SE05X is no longer applicable, use T1oI2C_UM11225 instead.
- *SE05x_CryptoModeSubType_t*
SE05x_CryptoModeSubType_t : :u8 renamed to SE05x_CryptoModeSubType_t : :union_8bit
- Endianness of ed25519 and mont_dh_25519 signature and shared secret are updated to little endian.

2.3.10 Examples / DEMO updates

Updated Examples:

- [Section 5.15 SE05X Get Info example](#) Updated to show CPLC data.

New Examples:

- [Section 5.24 Import External Object Prepare](#)
- [Section 5.25 Import External Object Create](#)
- [Section 5.27 Read object with Attestation](#)
- [Section 5.28 SE05X Transport Lock example](#)
- [Section 5.29 SE05X Transport UnLock example](#)
- [Section 5.30 SE05X Timestamp](#)
- [Section 5.38 Write APDU to buffer](#)

2.3.11 Documentation Changes

- Updated notes on `ssscli se05x reset` and `Se05x_API_DeleteAll_Iterative()`
- Updated documentation of SE05X layer of SSS APIs, e.g. `sss_se05x_key_store_load()` now mentions that this API does not do anything special on SE05X.
- Updated wifi-eap document.
- Changed logging styles and updated misc documentation with the same information.
- Added documentation for PKCS#11 standalone library
- Updated Greengrass documentation with new PKCS#11 project name
- Added documentation for Import External Object example
- Extended API Documentation for SE05X Low Level APIs

2.3.12 Other Miscellaneous Changes

- Bug fix: Remaining cache data and input data handled in sss_cipher_finish API
- OPC-UA Example enabled for compilation/running from Raspberry PI
- mbedTLS Upgraded to v02.16.02
- Added mbedCrypto for LPC55S / TF-M related work. (Ongoing, NXP Internal work)

2.4 Release v02.12.05

Handle PC Linux for VCOM Interface.

2.5 Release v02.12.04

Minor touch ups.

2.6 Release v02.12.03

AKM: Fix processing of keys without read policy.

2.7 Release v02.12.02

2.7.1 AKM Specific changes

- Fixed limitation in importing Key ID 00000001. (Using prefix 0xA5.)

2.7.2 Example Updates

- Updated example [Section 5.2.2 RSA Example](#) to also inject RSA key and run with FIPS compiled option.
- Added compile time check to skip examples [Section 5.2.4 HKDF Example](#) and [Section 5.2.7 ECDH Example](#) when compiled with FIPS

2.7.3 Other Miscellaneous Changes

- Added check to skip attempt to erase Trust Provisioned objects.

2.8 Release v02.12.01

2.8.1 AKM Specific changes

- Modified SE policy of keymaster HAL in Android
- Added *Section 5.19 SE05X Rotate PlatformSCP Keys Demo* to be build by Android build system.

2.8.2 New example / example updates

- Added support to retrieve existing certificates in pem format
- Ex `se05x_minimal.c` shows `kSE05x_MemoryType_PERSISTENT` memory. (Earlier it used `kSE05x_MemoryType_TRANSIENT_DESELECT`)
- Extended *Section 5.19 SE05X Rotate PlatformSCP Keys Demo* & it's documentation.

2.8.3 Other Miscellaneous Changes

- Updated Platform SCP Keys & SSD_NAME from Application Note
- Added *Using Platform SCP Keys from File System*
- Logging. Added `USE_COLORED_LOGS` inside `nxLog.c`
- Support FastSCP with and without KDF counter.
- Added support of VEN pin support for SE05X on Raspberry Pi / iMX6.
- Supports applet version 03.06.00

2.9 Release v02.12.00

2.9.1 New feature support

- Added demos *HMAC Example* and *ECDH Example*
- SHA (one shot and multi step) and HMAC (one shot and multi step) implementation in SCCP A71CH layer
- Added support for AWS Cloud example with Ease-of-Use configuration
- Added support of Multistep update for HMAC in Android Keymaster

2.9.2 SSSCLI / PyCLI Changes

- Updated AWS Provisioning certificate chain to remove Intermediate CA

2.9.3 Documentation Changes

- Added documentation for AWS Ease-of-Use

2.10 Release v02.11.03

2.10.1 File/Folder relocation

Moved this files: hostLib:

```
a71ch/src/a71ch_com.c => libCommon/infra/sm_connect.c
tstUtil/app_boot.h => libCommon/infra/app_boot.h
tstUtil/app_boot_nfc.c => libCommon/infra/app_boot_nfc.c
tstUtil/app_boot_nfc.h => libCommon/infra/app_boot_nfc.h
tstUtil/a7x_app_boot.c => libCommon/infra/sm_app_boot.c

tstUtil/tst_utils_kinetis.c => libCommon/infra/sm_demo_utils.c
tstUtil/tst_utils_kinetis.h => libCommon/infra/sm_demo_utils.h

tstUtil/tst_utils_rtos.c -> libCommon/infra/sm_demo_utils_rtos.c

inc/a71ch_const.h => sm_const.h
```

hostLib/libCommon/scp:

```
scp.c split to scp.c and scp_a7x.c
```

2.10.2 New feature support

- WiFi and cloud demos support for LPC55S

2.10.3 Scripts and Build changes

- I2C is not longer a valid define. Use SCI2C to select SCI2C for A71XX Family.
- NON Cmake based MCUXPresso projects from projects/ folder are removed and no longer supported. These examples are now separately released as KSDK Packages.

2.10.4 Documentation Changes

- Updated API usage description for T1oI2C protocol stack

2.10.5 Other Miscellaneous Changes

- PKCS#11 Testbench Integration

2.11 Internal Release v02.11.01

2.11.1 APIs & enum/types Changes

- Added `phNxpEse_data` in `iFrameInfo_t` also in `sFrameInfo_t`
- Changed order of parameters to `sss_channel_context_init`:

```
// Earlier:
    sss_status_t sss_channel_context_init(
        sss_session_t *session, sss_channel_t *context);

// Now: Parameters are swapped
    sss_status_t sss_channel_context_init(
        sss_channel_t *context, sss_session_t *session);
```

- Changed order of parameters to `sss_rng_context_init()`:

```
// Earlier:
    sss_status_t sss_rng_context_init(
        sss_session_t *session, sss_rng_context_t *context);

// Now: Parameters are swapped
    sss_status_t sss_rng_context_init(
        sss_rng_context_t *context, sss_session_t *session);
```

- Renamed `sss_channel_context_init` to `sss_tunnel_context_init`
- Renamed `sss_channel_context_free` to `sss_tunnel_context_free`

2.11.2 Functional Changes

- Added support for jrcp server on Android platform
- Fixed bug in VCOM Close for A71XX Family
- Handle WTX for CCID/PCSC Interface
- Added open62541 (OPC UA)
- Added RSA support to `<simw-top>/demos/linux/tls_client/scripts/tlsServer.sh`, `tlsSeClient.sh` and `tlsExtendedSeClient.sh` scripts.
- Added `/reset` support for jrcp server V1.

2.11.3 New feature support

- Added support for AWS Greengrass core on Rpi.

2.11.4 Scripts and Build changes

- Fixed MSVC Compiler warnings
- CMake option `SSS_HAVE_FIPS` is not a boolean. Earlier it was a string.
- AKM android based system set to build with `SSS_HAVE_FIPS` and `SSS_HAVE_SCP_SCP03_SSS`
- `SSS_HAVE_TESTCOUNTERPART` and `WithSSS_TestCounterPart` are deprecated. Please use `SSSFTR_SW_TESTCOUNTERPART`
- In file `hostLib/platform/imx/i2c_a7.c`, check `I2C_FUNC_SMBUS_READ_BLOCK_DATA` only for SCI2C (A71CH)
- Removed obsolete variables `ENGINE_DRIVEN_LIB_TYPE` and `BUILD_A71CH_OPENSSL_ENGINE` from cmake build scripts. Only the cmake option `WithSharedLIB` now determines the type of library (shared or static) that will be built / installed.
- AKM android based system set to build with All authentication types

2.11.5 Documentation Changes

- Fixed pycli pre-steps documentation for RaspberryPi.
- Updated documentation
- Added ssscli commands list.
- Added Ease-Of-Use documentation for Azure IoT Hub.

2.11.6 Communication Layer Changes

- Removed unwanted buffer handling from T1oI2C.

2.11.7 User Interface Changes

- Updated input parameters in `<simw-top>/demos/linux/tls_client/scripts/provisionTlsClient.py` script. Usage example:

```
python provisionTlsClient.py --key_type ecc --connection_data 169.254.0.1:8050
```

They are documented in [Section 6.1 Introduction on OpenSSL engine](#)

2.11.8 Other Miscellaneous Changes

- ECC Key overwrite on different curve id not allowed
- lwIP stack updated. Current version of lwIP is based on lwIP 2.1.2 and lwIP-contrib 2.1.0.
- Freertos version upgraded to 1.4.7_rev0
- Added NO_REGISTER_ALL flag for openssl engine (openssl 1.1.1) to use the capabilities specified in openssl conf file.
- openssl 1.1.1 warnings fixed.

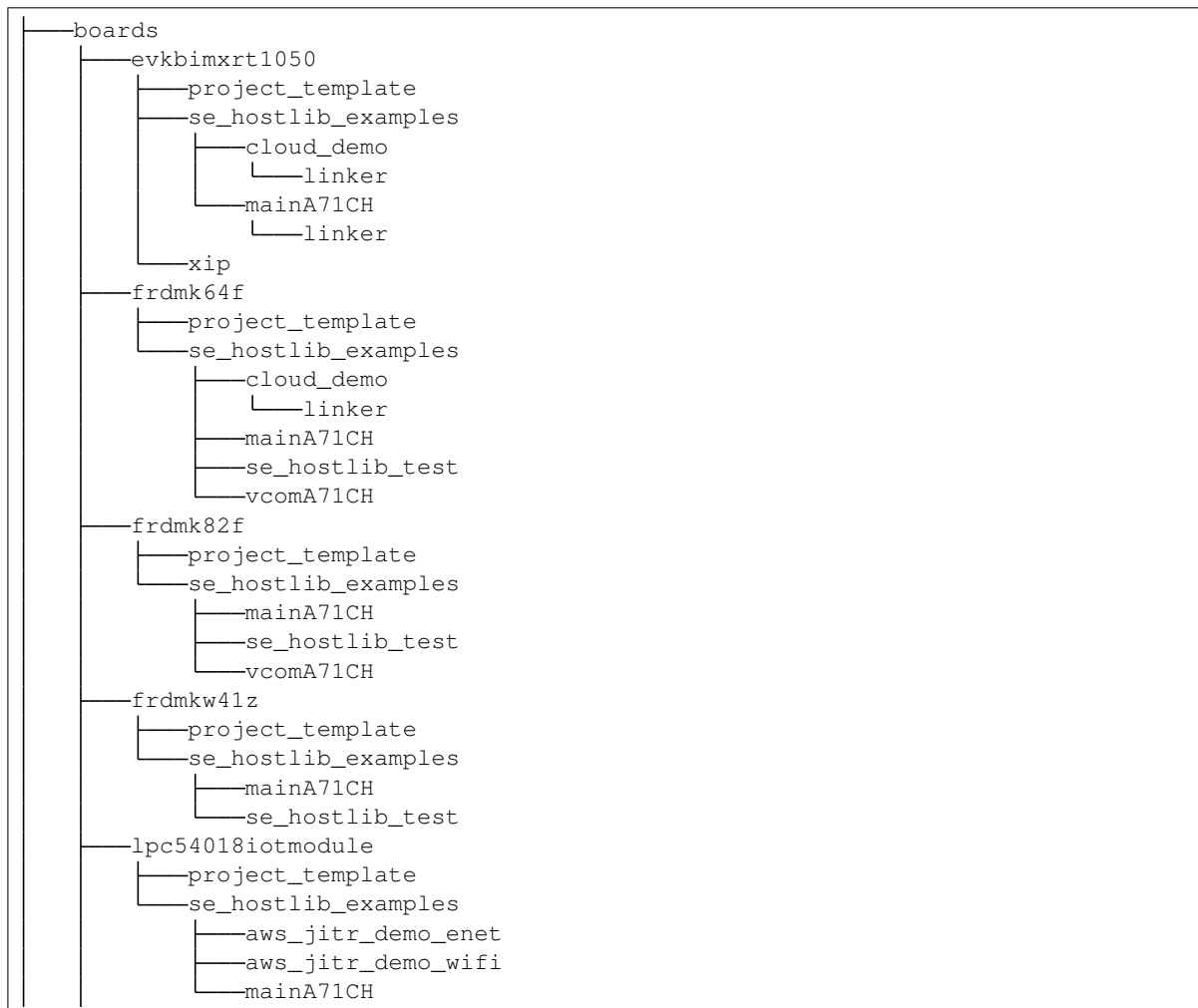
2.12 Release v02.11.00

2.12.1 File/Folder relocation

boards folder has been moved from

simw-top\ext\mcu-sdk to

simw-top\demos\ksdk\common



(continues on next page)

(continued from previous page)



freertos folder moved from

simw-top\ext\freertos to

simw-top\demos\ksdk\common\freertos



other boards folder has been moved from

simw-top\ext\boards to

simw-top\demos\ksdk\common\freertos\boards



2.12.2 APIs & enum/types Changes

- Removed `SE05x_AuthCtx_UserID_t`. Use `SE05x_AuthCtx_ID_t` instead
- Removed `SE_ConnType_t`. Use `SSS_Conn_Type_t` instead
- Removed `SEConnType_t`. Use `SSS_Conn_Type_t` instead
- Removed `AppletConfig_SM` from `SE05x_Applet_Feature_t`
- Renamed `se05x_TP_PlatformSCP03keys` to `se05x_RotatePlatformSCP03Keys`

2.12.3 Functional Changes

- **Added CCID/PCSC Interface (Experimental):**
 - Added `kType_SE_Conn_Type_PCSC`
 - Added weak function `SysTick_Handler_APP_CB()` to allow unblocking of threads. This is needed to handle the IRQ based design of CCID Middleware.
- **Added support for hmac-sha224 in mbedtls and openssl SSS MAC apis:**
 - Added `kAlgorithm_SSS_HMAC_SHA224`
- Fix a crash seen in `sss_mbedtls_mac_context_free`.
- Renamed and modified project `se05x_Get_UID` as `se05x_Get_Info` to include platform information also.

2.12.4 New platform support

- Added support for secure world and non secure world implementation of LPC55S.

2.12.5 Scripts and Build changes

- Added AOSP build support for Android keymaster.
- Added `Host=lpcpresso55s_s` and `Host=lpcpresso55s_ns` to support secure world implementation of LPC55S.
- No longer supporting `-DApplet_SE05X_Ver=02_02_00`
- Creating `cmake_options.mak` similar to `fsl_sss_ftr.h` so that customer build systems can be used/extended.

2.12.6 SSSCLI / PyCLI Changes

- Switched to Python 3
- Provisioning scripts refactored to be more consistent with internal variable names for keys and certificates. (No behaviour change)
- Updated cryptography patch to support BrainpoolP256R1 curve
- Refactored scripts to use different variable names.
- For some versions of python cryptography module, `key_size` was not available. Handling this within python library now.

- Added PKCS#12 format reference key creation.
- Enabled pcsc connection method

2.12.7 Documentation Changes

- Documentation for Demos updated. Earlier, KSDK demos were mentioning Raspberry Pi steps as well. This is removed now.
- Included documentation on how to get SE UID
- Included documentation for Ease of Use with IBM Watson and GCP
- Added default cmake options for imx and rpi build document.
- i.MX / Yocto instructions updated to include Python3 and func-timeout Python package

2.12.8 Communication Layer Changes

- Optimized T1oI2C transceive time by 3-8 ms.
- Poll waiting time has been reduced from 5ms to 1ms.
- T=1 I2C support for GP 0.39 specification.

Warning: You need to add both `-DT1oI2C` and `-DT1oI2C_UM1225_SE050` in your build system makefile to select T=1 over I2C Interface of SE050.

Earlier only `-DT1oI2C` was needed.

2.12.9 Other Miscellaneous Changes

- Support for DTLS example
- Included certificate chain in middleware for trust provisioned keys
- Included pre-built binary to get SE UID for FRDM-K64F, iMX-RT1050 and LPC55S
- Included pre-built binary for VCOM for LPC55S
- Added azure root certificate
- Key generation added for a71ch openssl engine (openssl 1.1.1)
- Compile time directives for SE050A/B/C in openssl engine
- Added `EX_SSS_BOOT_OPEN_HOST_SESSION` to let application decide on opening a host session
- Extended legacy openssl engine test scripts with multiple ecc keys testing. Also replaced ssscli tool with a71ch config tool for testing
- Added sampleConfig.json file for aws (linux) demo
- Added pre-built binaries to configure applet flavour (A, B or C) on iMX6 platform
- Added pre-built EXEs to configure applet flavour (A, B or C) from PC and VCOM Connection

2.13 Release v02.10.00

2.13.1 APIs & enum/types Changes

Renaming of structures to follow convention:

- `auth_scp03_dyn_context_t` -> `NXSCP03_DynCtx_t`
- `auth_scp03_static_context_t` -> `NXSCP03_StaticCtx_t`
- `se05x_auth_mech_scp03_context_t` -> `NXSCP03_AuthCtx_t`
- `se05x_auth_mech_fastscp_context_t` -> `SE05x_AuthCtx_FastScp_t`

Re-factored structure for cleaner isolation of static and dynamic objects:

- **SE05x_AuthCtx_FastScp_t, NXSCP03_AuthCtx_t**
 - Moved members to separate structure `NXFastSCP03_StaticCtx_t`

Re-factored structure to reclaim static memory when not needed:

- `NXSCP03_AuthCtx_t`, `SE05x_AuthCtx_FastScp_t` and `SE05x_AuthCtx_ID_t` now uses pointers to objects that can be freed eventually by the application after they are used during initial authentication.
- **a71ch_auth_context_t**
 - Removed `a71ch_auth_context_t` and using only `SE_Connect_Ctx_t`
- **sss_sscp_session**
 - Added member `mem_sscp_ctx : sscp_context_t` for memory.

Added configurability:

- `NXSCP03_StaticCtx_t` - Added `keyVersionNo` for platform SCP.
- **sss_se05x_channel_context_t**
 - Added member `channelLock : pthread_mutex_t` for simultaneous access.

Moved SE050 specific parameters to the end of structure:

- `NXSCP03_DynCtx_t` - `authType` is now the last member of the structure. Earlier it was the first structure.

New structure added:

- Added `sss_connect_ctx_t` and `SE_Connect_Ctx_t`

Promoted Auth to be generic instead of being SE050 Specific:

- `kSE05x_AuthType_UserID` -> `kSSS_AuthType_ID`
- `kSE05x_AuthType_SCP03` -> `kSSS_AuthType_SCP03`
- `kSE05x_AuthType_FastSCP` -> `kSSS_AuthType_FastSCP`
- `kSE05x_AuthType_AppletSCP03` -> `kSSS_AuthType_AppletSCP03`
- `kSE05x_AuthType_None` -> `kSSS_AuthType_None`

Add new use cases:

- `SmCommState_t` Added `skip_select_applet` to skip selecting the applet.

2.13.2 New platform support

- **Added LPC55s**
 - WiFi is not yet integrated on LPC55s demos
- **Raspberry PI**
 - Tested using Raspbian OS for SE050

2.13.3 Scripts and Build changes

- iMX RT 1050 drivers updated to SDK Release 2.6.1

2.13.4 SSSCLI / PyCLI Changes

- Added timeout mechanism in `sscli`

2.13.5 Other Miscellaneous Changes

- In few files, NON-ASCII characters are replaced with their ASCII equivalents.

2.14 Release v02.09.00

- Integration Applet release v02.05.00
- Renamed `hostLib/se05x_02_02_00` to `hostLib/se05x_02_02_xx`
- No longer using the term PIN. Using UserID for Authentication Objects
- Integration Applet release v02.03.00

2.14.1 Middleware

- Refactor: Changed `kSSS_KeyType_Certificate` to `kSSS_KeyType_Binary`

2.14.2 CLI Tool

- Updated `sscli --help` documentation
- option `--format` has been added to CLI commands to specify file format.
- option `--hashalgo` has been added to sign and verify commands.
- option `--auth_type` has been added to connect command.

2.15 Release v02.07.00

2.15.1 Middleware

- Re-Wrote CMake build system to use Drop Down for selection of values instead of Check Boxes.
- WithSMCOM_SOCKET renamed to WithSMCOM_JRCP_V1
- WithSMCOM_JRCP renamed to WithSMCOM_JRCP_V2
- Added support for Applet_SE05X_Ver_02_02_xx, removed support for other version of Applet.
- Renamed eSE05xTAG_t to SE05x_TAG_t
- Added Negative test cases to check T1oI2C functionality.
- Hard reset recovery Mechanism implemented.
- Logging implemented for Android platform

2.15.2 SMCom

- Added Thread smCom Layer (Experimental)
- Changed T=1 over I2C layer to support JCOP SR6 (Handling of CRC Changed)

Note: This change makes it in-compatible with older CES release of SE050.

- API exposed for clearing T1oI2C 7816 params to SMCom Layer (smComT1oI2C_ComReset())
- Hard Resetting IC using ENA pin is implemented.
- Read polling count decreases.

2.15.3 SSS APIs

- Refactored RSA/CRT RSA Key types. RSA Deemed RAW By default.
- Added separate policies for Symmetric / Asymmetric keys
- Added policies for counters

2.15.4 CLI Tool

- Changed `set ecc NIST_P256` to `set ecc`
- Session Open uses PIN with pin ID “0x7DA00001” and key “[0xC0, 0x01, 0x02, 0x03, 0x04]”
- CLI Failure Stack dump captured in log file.
- Added ‘se05x’ specific commands

2.15.5 Development Platform

- (Hikey960) Added support for SE050 v2.2.4
- (Hikey960) Support for HMAC
- (Hikey960) Support for HMAC sign/verify
- (Hikey960) Support for AES encrypt/decrypt
- (Hikey960) Support for Attestation

2.16 Release v02.06.00

2.16.1 Demos

2.16.2 Plug & Trust Middleware

- (SE050) Added basic policies
- (SE050) Updated enums for EC Curve IDs
- (SE050) Support v1.1.0 and v1.2.0 from same stack
- (SSS, SE050) Added policies
- (CLI-Tool) Changed `ecc 256` to `ecc NIST_P256`

2.16.3 Documentation

- Updated section on *Logging*
- Updated section on *AOSP build Environment Setup*
- Added/Updated CMake Options
- Updated navbar and searchbox placement for handling of html display

2.16.4 Development Platform

- (Hikey960) Added support for SE050 v1.2.0
- (Hikey960) Support for ECC-521 and AES-128

2.17 Release v02.05.00

Added Android Key Master.

2.18 Release v02.04.00

Added SE050 EAR CH and SE050 EAR CL Applet supports.

2.19 Release 02.03.00

Added support for SSS APIs.

PLUG & TRUST MW STACK

3.1 Features

Supported development platforms: *Development Platforms*

The Plug & Trust Middleware Stack supports:

- **Directly accessible standard API/integration**
 - mbed TLS
 - OpenSSL
 - Android KeyStore
- **Examples for quick integration**
 - TLS examples
 - Cloud service on-boarding examples
- Integration with NXP EdgeLock 2GO cloud service (Upcoming)

3.2 Plug & Trust MW : Block Diagram



3.3 SSS APIs

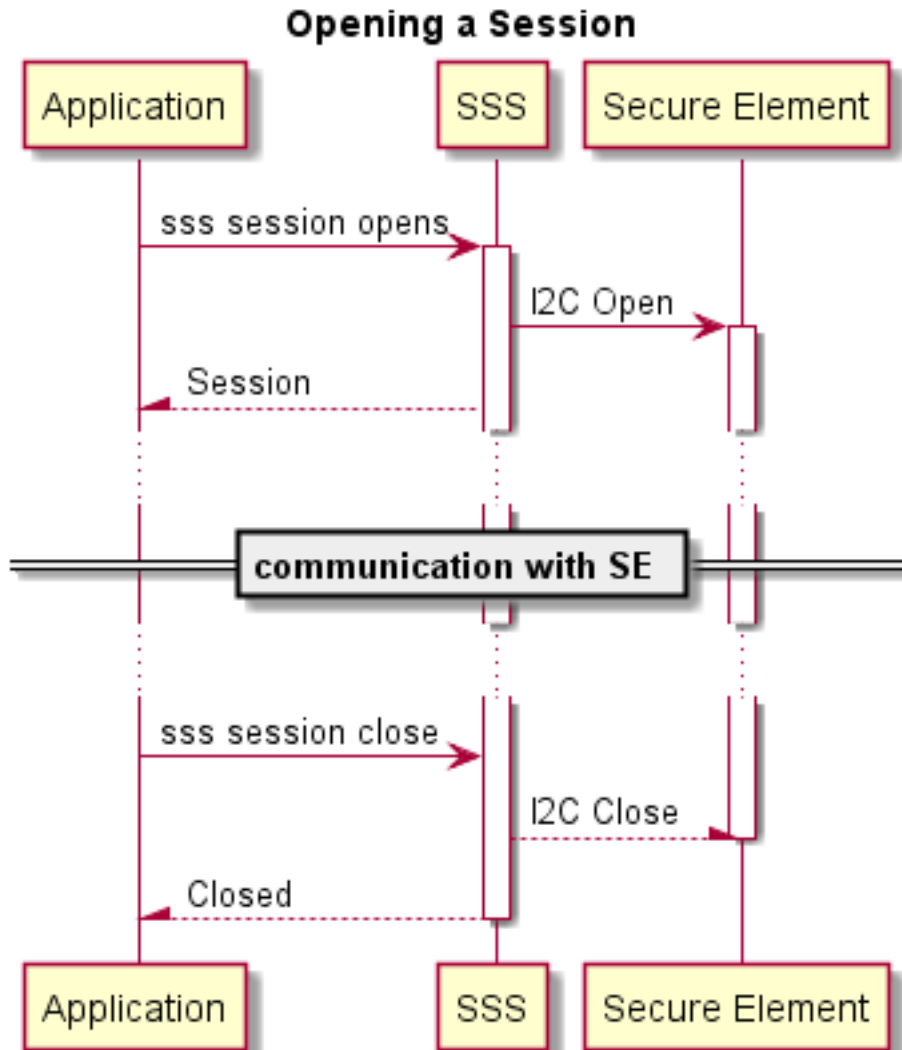
3.3.1 SSS: Introduction

SSS is an acronym for **Secure Sub System**

The SSS APIs are functional APIs to abstract the access to various types of Cryptographic Sub Systems. Such secure subsystems could be (but not limited to):

- Secure Elements
- Secure Enclaves
- Cryptographic HWs

3.3.2 Session



See `sss_session_open()`, `sss_session_close()`

Opening a Session

Sessions are tightly coupled with underlying system. For opening a session, `sss_session_open()`, subsystem is passed from `sss_type_t`, while the parameter `connectionData` plays a pivotal role where there are subsystem specific parameters to be handled.

SE050 Session

For example, a dedicated `SE_Connect_Ctx_t` is passed while opening a session to the SE050 Secure Element.

```
/* Opening a password/user id based session */
sss_session_t session      = {0};
SE_Connect_Ctx_t connectCtx = {0};
sss_object_t ex_id         = {0}; /* Object to store the id value */
sss_status_t status;

/* we need a host session and key store to access
 * values from host */
sss_session_t host_session  = {0};
sss_key_store_t host_keystore = {0};

/* Value which we are going to use eventually.
 * We will not use it directly, but indirectly.
 * We will store this in in ex_id
 */
const uint8_t value_user_id[] = EX_SSS_AUTH_SE05X_UserID_VALUE;

/* Open host session and key store */
status = sss_session_open(&host_session, kType_SSS_Software, 0, kSSS_ConnectionType_
↳ Plain, NULL);
TEST_ASSERT_EQUAL_HEX(kStatus_SSS_Success, status);
status = sss_key_store_context_init(&host_keystore, &host_session);
TEST_ASSERT_EQUAL_HEX(kStatus_SSS_Success, status);

/* Set the auth object in Connect context */
connectCtx.auth.ctx.idobj.pObj = &ex_id;
status                          = sss_key_object_init(connectCtx.auth.ctx.idobj.pObj, &
↳ host_keystore);
TEST_ASSERT_EQUAL_HEX(kStatus_SSS_Success, status);

/* On the host, allocate object and set value */
status = sss_key_object_allocate_handle(connectCtx.auth.ctx.idobj.pObj,
    __LINE__,
    kSSS_KeyPart_Default,
    kSSS_CipherType_UserID,
    sizeof(value_user_id),
    kKeyObject_Mode_Transient);
TEST_ASSERT_EQUAL_HEX(kStatus_SSS_Success, status);
status = sss_key_store_set_key(&host_keystore,
    connectCtx.auth.ctx.idobj.pObj,
    value_user_id,
    sizeof(value_user_id),
    8 * sizeof(value_user_id),
    NULL,
    0);
TEST_ASSERT_EQUAL_HEX(kStatus_SSS_Success, status);

/* Now we connect to the SE. In this example, we
 * are connecting to over JRCP V2 interface */
connectCtx.portName      = gsza71SocketPortDefault;
connectCtx.connType      = kType_SE_Conn_Type_JRCP_V2;
connectCtx.auth.authType = kSSS_AuthType_ID;
```

(continues on next page)

(continued from previous page)

```

const uint32_t authObjectIdForPasswordSession = EX_SSS_AUTH_SE05X_UserID_AUTH_ID;

/* Open the session to the secure element */
status = sss_session_open(
    &session, kType_SSS_SecureElement, authObjectIdForPasswordSession, kSSS_
    ↳ConnectionType_Password, &connectCtx);
TEST_ASSERT_EQUAL_HEX(kStatus_SSS_Success, status);
/* .... operations with the SE .... */
/* Close the connection to secure element */
sss_session_close(&session);

```

APIS

group **sss_session**
Manage session.

Enums

enum sss_session_prop_au8_t

Properties of session that are S32

From 0 to kSSS_SessionProp_Optional_Prop_Start, around $2^{24} = 16777215$ Properties are possible.

From 0 to kSSS_SessionProp_Optional_Prop_Start, around $2^{24} = 16777215$ Properties are possible.

Values:

kSSS_SessionProp_au8_NA = 0
Invalid

kSSS_SessionProp_szName
Name of the product, string

kSSS_SessionProp_UID
Unique Identifier

kSSS_SessionProp_au8_Optional_Start = 0x00FFFFFFu
Optional Properties Start

kSSS_SessionProp_au8_Proprietary_Start = 0x01FFFFFFu
Proprietary Properties Start

enum sss_session_prop_u32_t

Properties of session that are U32

From 0 to kSSS_SessionProp_Optional_Prop_Start, around $2^{24} = 16777215$ Properties are possible.

From 0 to kSSS_SessionProp_Optional_Prop_Start, around $2^{24} = 16777215$ Properties are possible.

Values:

kSSS_SessionProp_u32_NA = 0
Invalid

kSSS_SessionProp_VerMaj
Major version

kSSS_SessionProp_VerMin
Minor Version

kSSS_SessionProp_VerDev
Development Version

kSSS_SessionProp_UIDLen

kSSS_SessionProp_u32_Optional_Start = 0x00FFFFFFu
Optional Properties Start

kSSS_KeyStoreProp_FreeMem_Persistent
How much persistent memory is free

kSSS_KeyStoreProp_FreeMem_Transient
How much transient memory is free

kSSS_SessionProp_u32_Proprietary_Start = 0x01FFFFFFu
Proprietary Properties Start

Functions

void **sss_session_close** (*sss_session_t* *session)
Close session between application and security subsystem.

This function closes a session which has been opened with a security subsystem. All commands within the session must have completed before this function can be called. The implementation must do nothing if the input *session* parameter is NULL.

Parameters

- *session*: Session context.

sss_status_t **sss_session_create** (*sss_session_t* *session, *sss_type_t* subsystem, uint32_t application_id, *sss_connection_type_t* connection_type, void *connectionData)

Same as *sss_session_open* but to support sub systems that explicitly need a create before opening.

For the sake of portability across various sub systems, the applicaiton has to call *sss_session_create* before calling *sss_session_open*.

Parameters

- [inout] *session*: Pointer to session context
- [in] *subsystem*: See *sss_session_open*
- [in] *application_id*: See *sss_session_open*
- [in] *connection_type*: See *sss_session_open*
- [in] *connectionData*: See *sss_session_open*

void **sss_session_delete** (*sss_session_t* *session)
Counterpart to *sss_session_create*

Similar to constraint on *sss_session_create*, application may call *sss_session_delete* to explicitly release all underlying/used session specific resoures of that implementation.

sss_status_t **sss_session_open** (*sss_session_t* *session, *sss_type_t* subsystem, uint32_t application_id, *sss_connection_type_t* connection_type, void *connectionData)

Open session between application and a security subsystem.

Open virtual session between application (user context) and a security subsystem and function thereof. Pointer to session shall be supplied to all SSS APIs as argument. Low level SSS functions can provide implementation specific behaviour based on the session argument.

Return status

Parameters

- [inout] session: Session context.
- [in] subsystem: Indicates which security subsystem is selected to be used.
- [in] application_id: ObjectId/AuthenticationID Connecting to:
 - application_id == 0 => Super use / Platform user
 - Anything else => Authenticated user
- [in] connection_type: How are we connecting to the system.
- [inout] connectionData: subsystem specific connection parameters.

sss_status_t **sss_session_prop_get_au8** (*sss_session_t* *session, uint32_t property, uint8_t *pValue, size_t *pValueLen)

Get an underlying property of the crypto sub system.

This API is used to get values that are numeric in nature.

Property can be either fixed value that is calculated at compile time and returned directly, or it may involve some access to the underlying system.

Return

Parameters

- [in] session: Session context
- [in] property: Value that is part of *sss_session_prop_au8_t*
- [out] pValue: Output buffer array
- [inout] pValueLen: Count of values there are/must be read

sss_status_t **sss_session_prop_get_u32** (*sss_session_t* *session, uint32_t property, uint32_t *pValue)

Get an underlying property of the crypto sub system.

This API is used to get values that are numeric in nature.

Property can be either fixed value that is calculated at compile time and returned directly, or it may involve some access to the underlying system.

For applicable properties see *sss_session_prop_u32_t*

Return

Parameters

- [in] session: Session context
- [in] property: Value that is part of *sss_session_prop_u32_t*
- [out] pValue:

```
struct sss_session_t
#include <fsl_sss_api.h> Root session.
```

This is a *singleton* for each connection (physical/logical) to individual cryptographic system.

Public Members

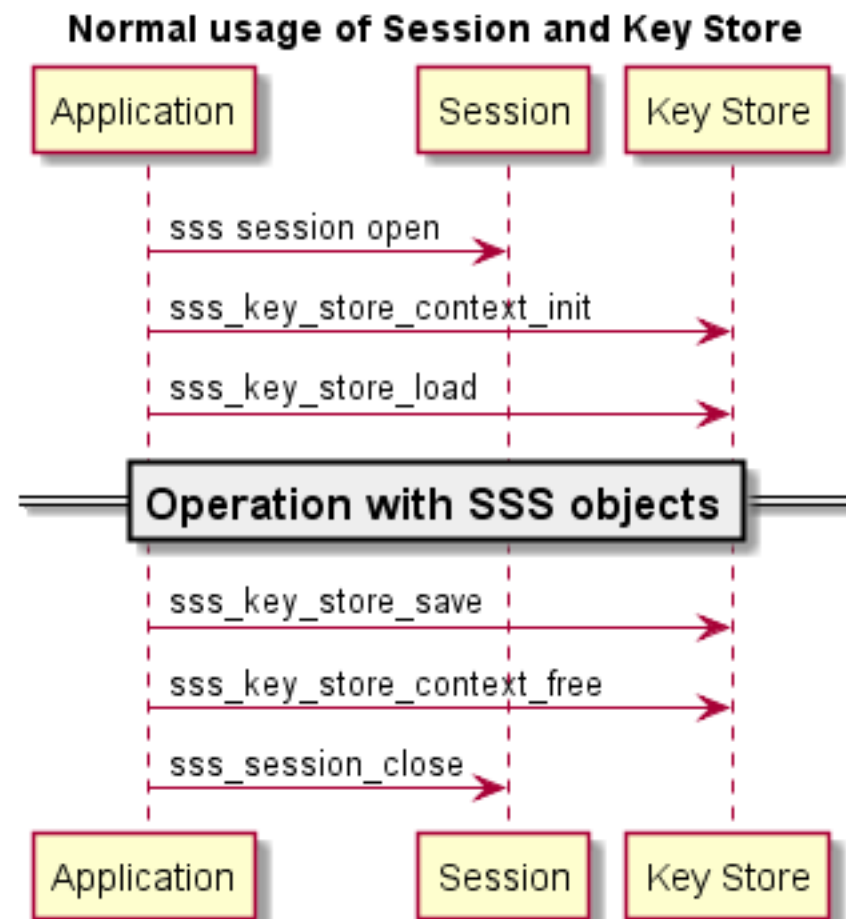
```
uint8_t data[(0 + (1 * sizeof(void *)) + (1 * sizeof(void *)) + (8 * sizeof(void *)) + 16)]
```

```
struct sss_session_t::[anonymous] extension
Reserved memory for implementation specific extension
```

```
sss_type_t subsystem
Indicates which security subsystem is selected.
This is set when sss_session_open is successful
```

3.3.3 Key Store

KeyStore is a container for all secure keys and objects inside a secure storage.



APIs

group **sss_key_store**

Secure storage for keys and certificates.

Enums

enum **sss_key_store_prop_au8_t**

properties of a Key Store that return array

Values:

kSSS_KeyStoreProp_au8_Optional_Start = 0x00FFFFFFu

Optional Properties Start

enum **sss_tunnel_dest_t**

Entity on the other side of the tunnel

Values:

kSSS_TunnelDest_None = 0

Default value

kSSS_TunnelType_Se05x_Iot_applet

SE05X IoT Applet

Functions

sss_status_t **sss_key_store_allocate** (*sss_key_store_t* *keyStore, uint32_t keyStoreId)

Get handle to key store. If the key store already exists, nothing is allocated. If the key store does not exist, new empty key store is created and initialized. Key store context structure is updated with actual information.

Parameters

- [out] keyStore: Pointer to key store context. Key store context is updated on function return.
- keyStoreId: Implementation specific ID, can be used in case security subsystem manages multiple different key stores.

void **sss_key_store_context_free** (*sss_key_store_t* *keyStore)

Destructor for the key store context.

sss_status_t **sss_key_store_context_init** (*sss_key_store_t* *keyStore, *sss_session_t* *session)

Constructor for the key store context data structure.

Parameters

- [out] keyStore: Pointer to key store context. Key store context is updated on function return.
- session: Session context.

sss_status_t **sss_key_store_erase_key** (*sss_key_store_t* *keyStore, *sss_object_t* *keyObject)

Delete / destroy allocated keyObject .

Return The sss status.

Parameters

- `keyStore`: The key store
- `keyObject`: The key object to be deleted

sss_status_t **sss_key_store_freeze_key** (*sss_key_store_t* *keyStore, *sss_object_t* *keyObject)

The referenced key cannot be updated any more.

Return The sss status.

Parameters

- `keyStore`: The key store
- `keyObject`: The key object to be locked / frozen.

sss_status_t **sss_key_store_generate_key** (*sss_key_store_t* *keyStore, *sss_object_t* *keyObject, *size_t* keyBitLen, void *options)

This function generates key[] in the destination key store.

sss_status_t **sss_key_store_get_key** (*sss_key_store_t* *keyStore, *sss_object_t* *keyObject, *uint8_t* *data, *size_t* *dataLen, *size_t* *pKeyBitLen)

This function exports plain key[] from key store (if constraints and user id allows reading)

sss_status_t **sss_key_store_load** (*sss_key_store_t* *keyStore)

Load from persistent memory to cached objects.

sss_status_t **sss_key_store_open_key** (*sss_key_store_t* *keyStore, *sss_object_t* *keyObject)

Access key store using one more level of encryption.

e.g. Access keys / encryption key during storage

Return The sss status.

Parameters

- `keyStore`: The key store
- `keyObject`: The key object that is to be used as a KEK (Key Encryption Key)

sss_status_t **sss_key_store_save** (*sss_key_store_t* *keyStore)

Save all cached persistent objects to persistent memory.

sss_status_t **sss_key_store_set_key** (*sss_key_store_t* *keyStore, *sss_object_t* *keyObject, **const** *uint8_t* *data, *size_t* dataLen, *size_t* keyBitLen, void *options, *size_t* optionsLen)

This function moves data[] from memory to the destination key store.

Return

Parameters

- `keyStore`: Key store context
- `keyObject`: Reference to a key and it's properties
- `data`: Data to be stored in Key. When setting ecc private key only, do not include key header.
- `dataLen`: Length of the data
- `keyBitLen`: Crypto algorithm key bit length
- `options`: Pointer to implementation specific options
- `optionsLen`: Length of the options in bytes

```
struct sss_key_store_t
```

#include <fsl_sss_api.h> Store for secure and non secure key objects within a cryptographic system.

- A cryptographic system may have more than partitions to store such keys.

Public Members

```
uint8_t data[(0 + (1 * sizeof(void *)) + (4 * sizeof(void *)) + 16)]
```

```
struct sss_key_store_t::[anonymous] extension
```

Reserved memory for implementation specific extension

```
sss_session_t *session
```

Virtual connection between application (user context) and specific security subsystem and function thereof.

Key Format

The `sss_key_store_set_key` and `sss_key_store_get_key` API's do not impose a specific format on the data parameter. Different implementations of the SSS API can have different capabilities in dealing with an input format (relevant for `sss_key_store_set_key`) and will use a specific output format (relevant for `sss_key_store_get_key`). The following section illustrates this by taking the example of the SE050 implementation in the context of EC Key pairs.

EC Key pair

When passing an EC key pair as data argument to the `sss_key_store_set_key` API, the key pair data must be DER encoded using either the pkcs#8 format or classic OpenSSL format.

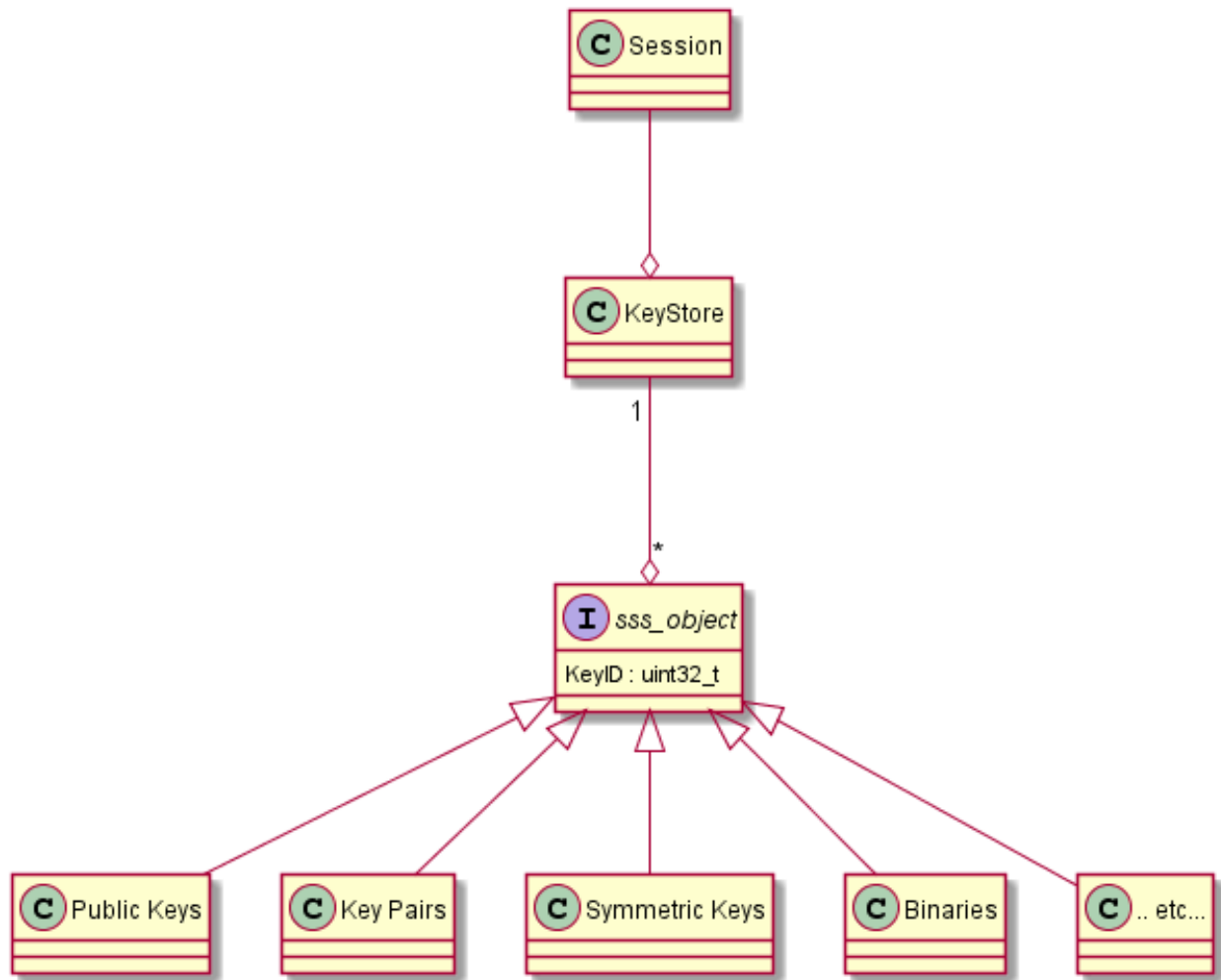
When retrieving an EC key pair as data argument from the `sss_key_store_get` API, the full key pair cannot be retrieved. Instead the public key value is returned. The public key is retrieved in ANSI X9.62 uncompressed format.

3.3.4 Key Object

Objects / Key Objects are Low level entities of key/certificates in SSS domain.

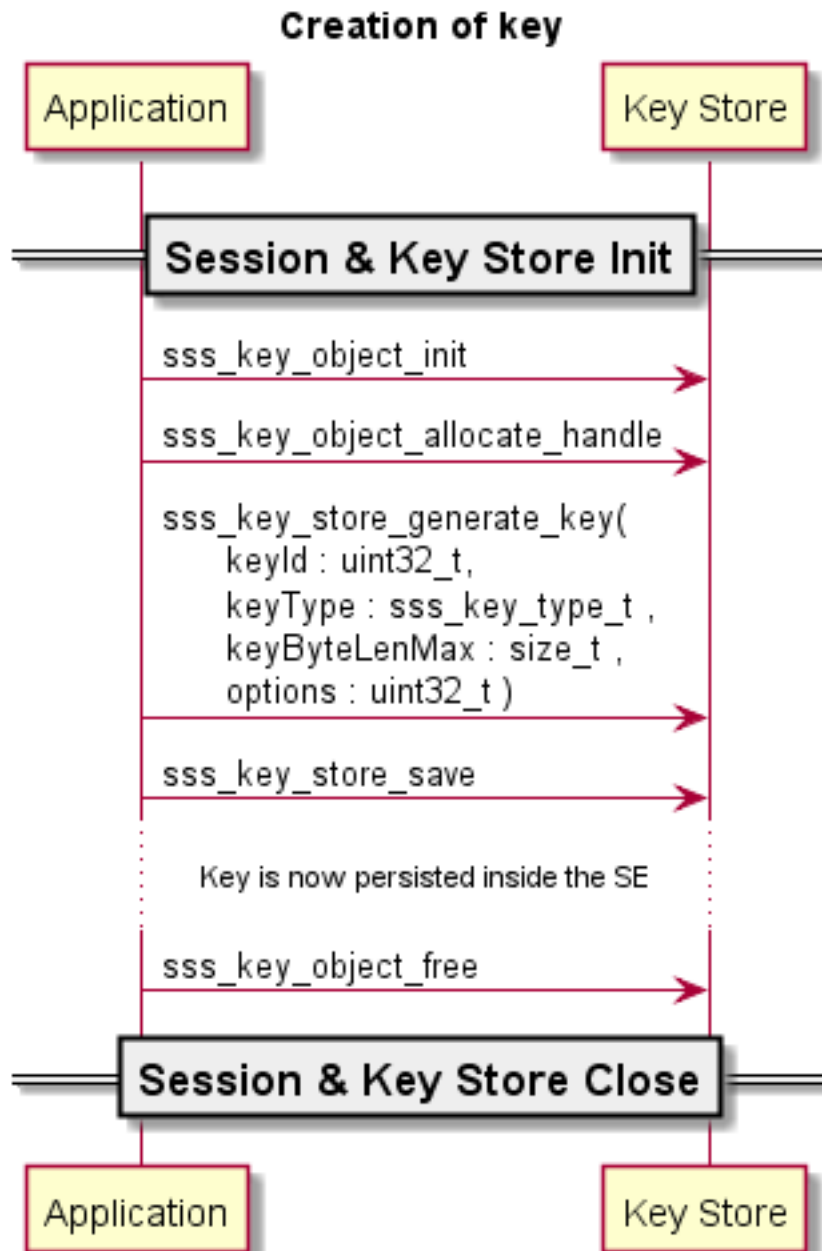
Below we can see UML Hierarchy of an object:

Container : Session / Key Store / Objects



Create / Provision

To create a key, the sequence of APIs looks as under. This is generally done during provisioning stage.



To set (inject) values in a previously allocated key, the sequence of APDUs look as under.

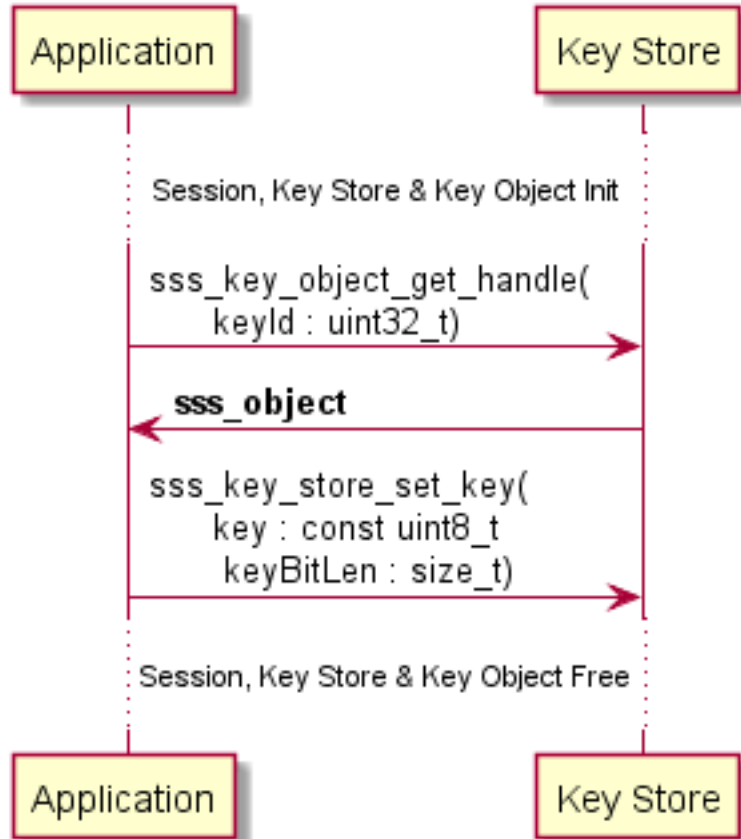
Note: Policies

This section would be updated later to show case creation of keys with different policies attached to it.

Change value of previously created Objects

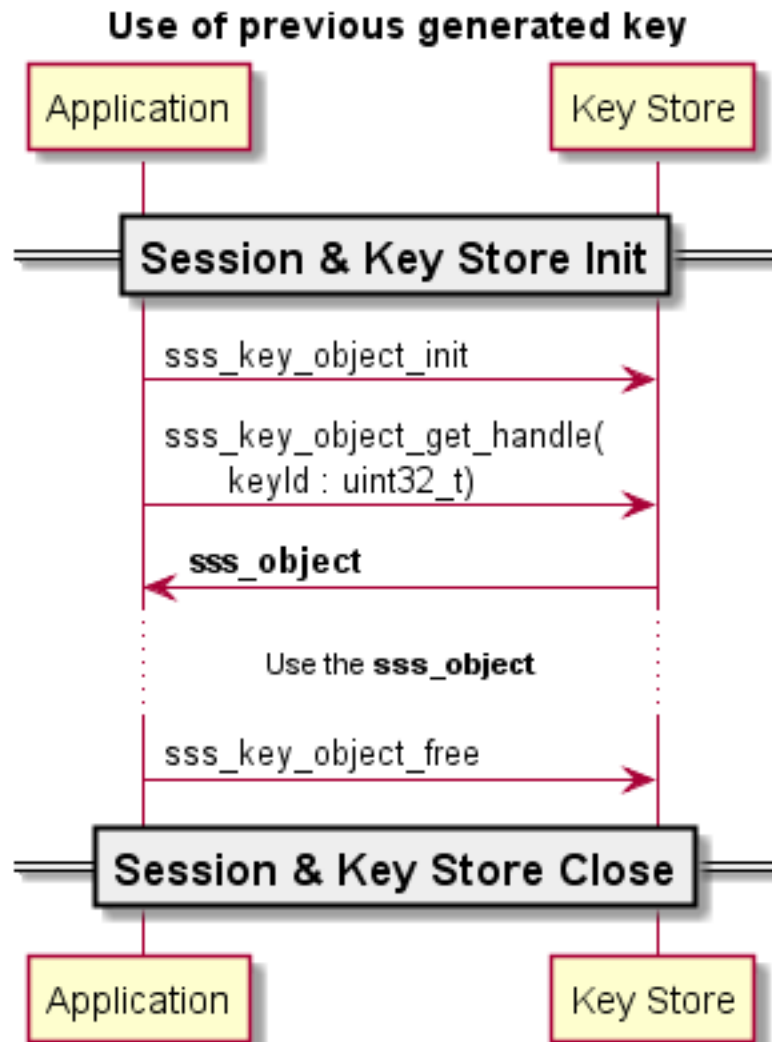
To create a key, the sequence of APIs looks as under:

Change value of previously created object



Use previously provisioned/created Keys/Objects

To use a key, the API sequence is as under:



APIs

group **sss_key_object**

Low level iota of key/certificates in SSS domain.

Functions

sss_status_t **sss_key_object_allocate_handle** (*sss_object_t* *keyObject, uint32_t keyId, *sss_key_part_t* keyPart, *sss_cipher_type_t* cipherType, size_t keyByteLenMax, uint32_t options)

Allocate / pre-provision memory for new key.

This API allows underlying cryptographic subsystems to perform preconditions of before creating any cryptographic key object.

Return Status of object allocation.

Parameters

- [inout] `keyObject`: The object If required, update implementation defined values inside the `keyObject`
- `keyId`: External Key ID. Later on this may be used by [sss_key_object_get_handle](#)
- `keyPart`: See [sss_key_part_t](#)
- `cipherType`: See [sss_cipher_type_t](#)
- `keyByteLenMax`: Maximum storage this type of key may need. For systems that have their own internal allocation table this would help
- `options`: 0 = Persistent Key (Default) or Transient Key. See [sss_key_object_mode_t](#)

void **sss_key_object_free** ([sss_object_t](#) *`keyObject`)

Destructor for the key object. The function frees key object context.

Parameters

- `keyObject`: Pointer to key object context.

[sss_status_t](#) **sss_key_object_get_access** ([sss_object_t](#) *`keyObject`, [uint32_t](#) *`access`)

Check what are access restrictions on an object

Return

Parameters

- `keyObject`: Object
- `access`: What is permitted

[sss_status_t](#) **sss_key_object_get_handle** ([sss_object_t](#) *`keyObject`, [uint32_t](#) `keyId`)

Get handle to an existing allocated/provisioned/created Object.

See [sss_key_object_allocate_handle](#).

After calling this API, Ideally `keyObject` should become equivalent to as set after the calling of [sss_key_object_allocate_handle](#) api.

Return The sss status.

Parameters

- `keyObject`: The key object
- [in] `keyId`: The key identifier

[sss_status_t](#) **sss_key_object_get_purpose** ([sss_object_t](#) *`keyObject`, [sss_mode_t](#) *`purpose`)

Check what is purpose restrictions on an object

Return

Parameters

- `keyObject`: Object to be checked
- `purpose`: Know what is permitted.

[sss_status_t](#) **sss_key_object_get_user** ([sss_object_t](#) *`keyObject`, [uint32_t](#) *`user`)

get attributes

sss_status_t **sss_key_object_init** (*sss_object_t* *keyObject, *sss_key_store_t* *keyStore)

Constructor for a key object data structure. The function initializes keyObject data structure and associates it with a key store in which the plain key and other attributes are stored.

Return Status of the operation

Parameters

- keyObject:
- keyStore:

Return Value

- *kStatus_SSS_Success*: The operation has completed successfully.
- *kStatus_SSS_Fail*: The operation has failed.
- *kStatus_SSS_InvalidArgument*: One of the arguments is invalid for the function to execute.

sss_status_t **sss_key_object_set_access** (*sss_object_t* *keyObject, *uint32_t* access, *uint32_t* options)

Assign access permissions to a key object.

Parameters

- keyObject: the object where permission restrictions are applied
- access: Logical OR of read, write, delete, use, change attributes defined by enum *_sss_access_permission*.
- options: Transient or persistent update. Allows for transient update of persistent attributes.

sss_status_t **sss_key_object_set_eccgfp_group** (*sss_object_t* *keyObject, *sss_eccgfp_group_t* *group)

Set elliptic curve domain parameters over Fp for a key object.

When the key object is a reference to one of ECC Private, ECC Public or ECC Pair key types, this function shall be used to specify the exact domain parameters prior to using the key object for ECDSA or ECDH algorithms.

Parameters

- keyObject: The destination key object
- group: Pointer to elliptic curve domain parameters over Fp (sextuple p,a,b,G,n,h)

sss_status_t **sss_key_object_set_purpose** (*sss_object_t* *keyObject, *sss_mode_t* purpose, *uint32_t* options)

Assign purpose to a key object.

Parameters

- keyObject: the object where permission restrictions are applied
- purpose: Usage of the key.
- options: Transient or persistent update. Allows for transient update of persistent attributes.

sss_status_t **sss_key_object_set_user** (*sss_object_t* *keyObject, *uint32_t* user, *uint32_t* options)

Assign user to a key object.

Parameters

- `keyObject`: the object where permission restrictions are applied
- `user`: Assign User id for a key object. The user is kept in the key store along with the key data and other properties.
- `options`: Transient or persistent update. Allows for transient update of persistent attributes.

struct `sss_object_t`

#include <fsl_sss_api.h> An object (secure / non-secure) within a Key Store.

Public Members

`uint32_t cipherType`

cipherType type from *sss_cipher_type_t*

`uint8_t data[(0 + (1 * sizeof(void *))) + (2 * sizeof(int)) + (4 * sizeof(void *)) + 16]`

struct *sss_object_t*::[anonymous] extension

Reserved memory for implementation specific extension

`uint32_t keyId`

Application specific key identifier. The keyId is kept in the key store along with the key data and other properties.

sss_key_store_t ***keyStore**

key store holding the data and other properties

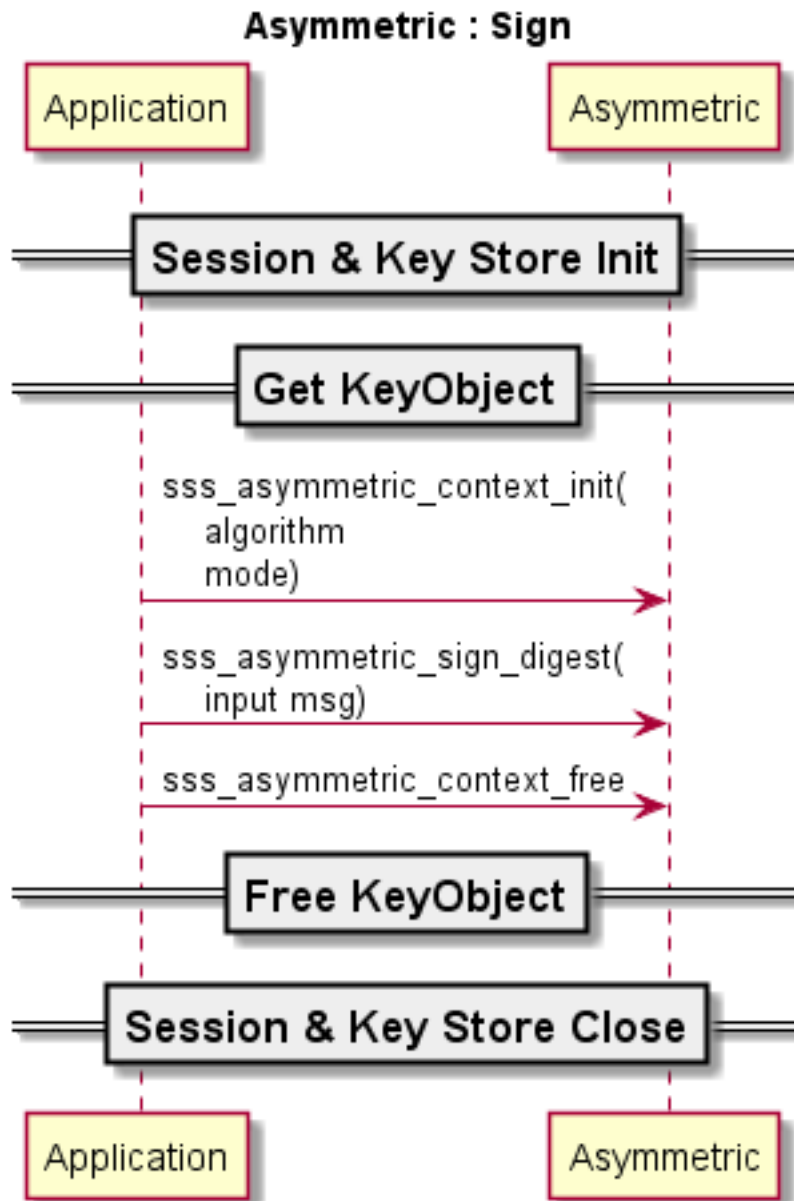
`uint32_t objectType`

The type/part of object is refernced from *sss_key_part_t*

3.3.5 Asymmetric

Sign

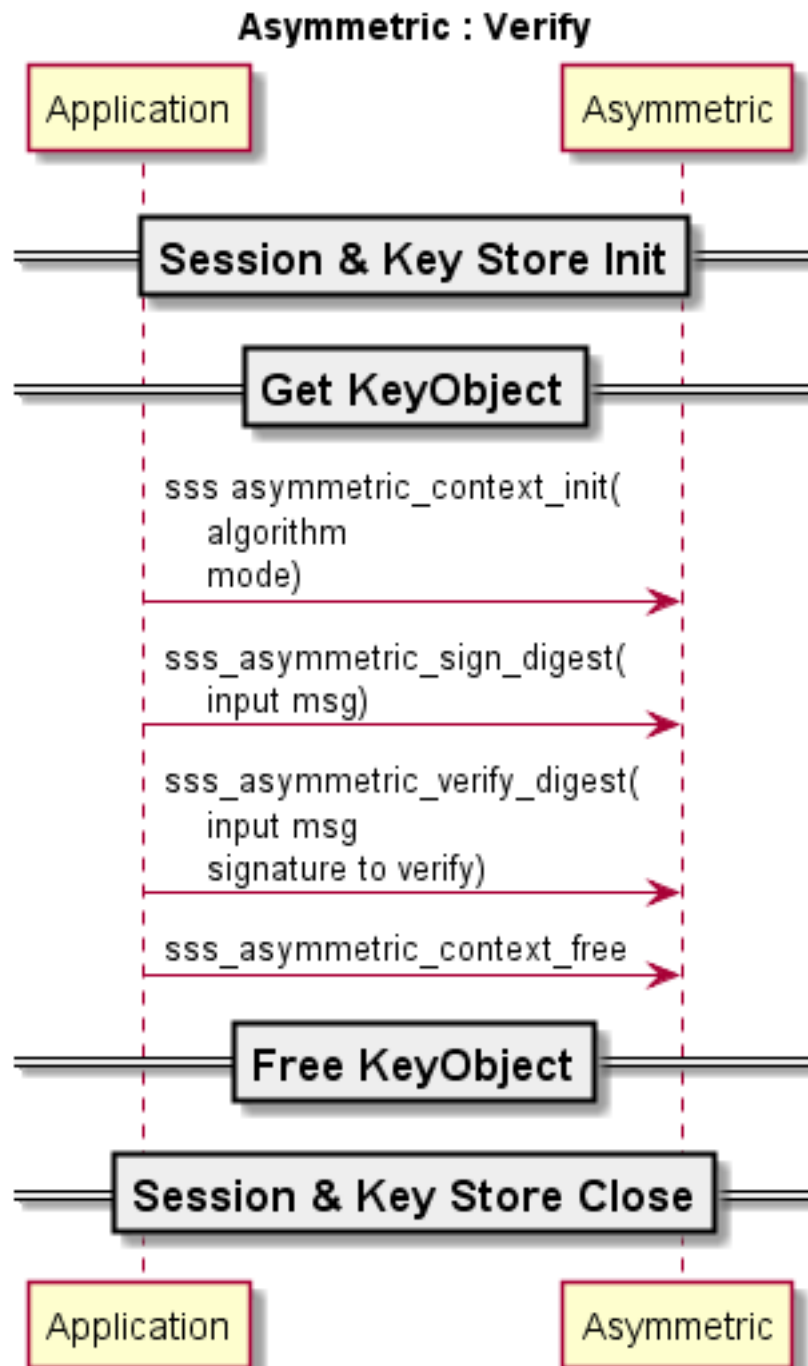
To perform sign operation , the sequence of APIs looks as under.

**Note:**

- 1) To perform rsa sign and verify on plain data (with hash calculated inside SE), use `sss_se05x_asymmetric_sign` and `sss_se05x_asymmetric_verify` apis.
- 2) Sign / Verify operations with Twisted Edward curve is supported only on plain data with hash calculated inside SE. Use `sss_se05x_asymmetric_sign` and `sss_se05x_asymmetric_verify` apis. Only SHA512 is supported.

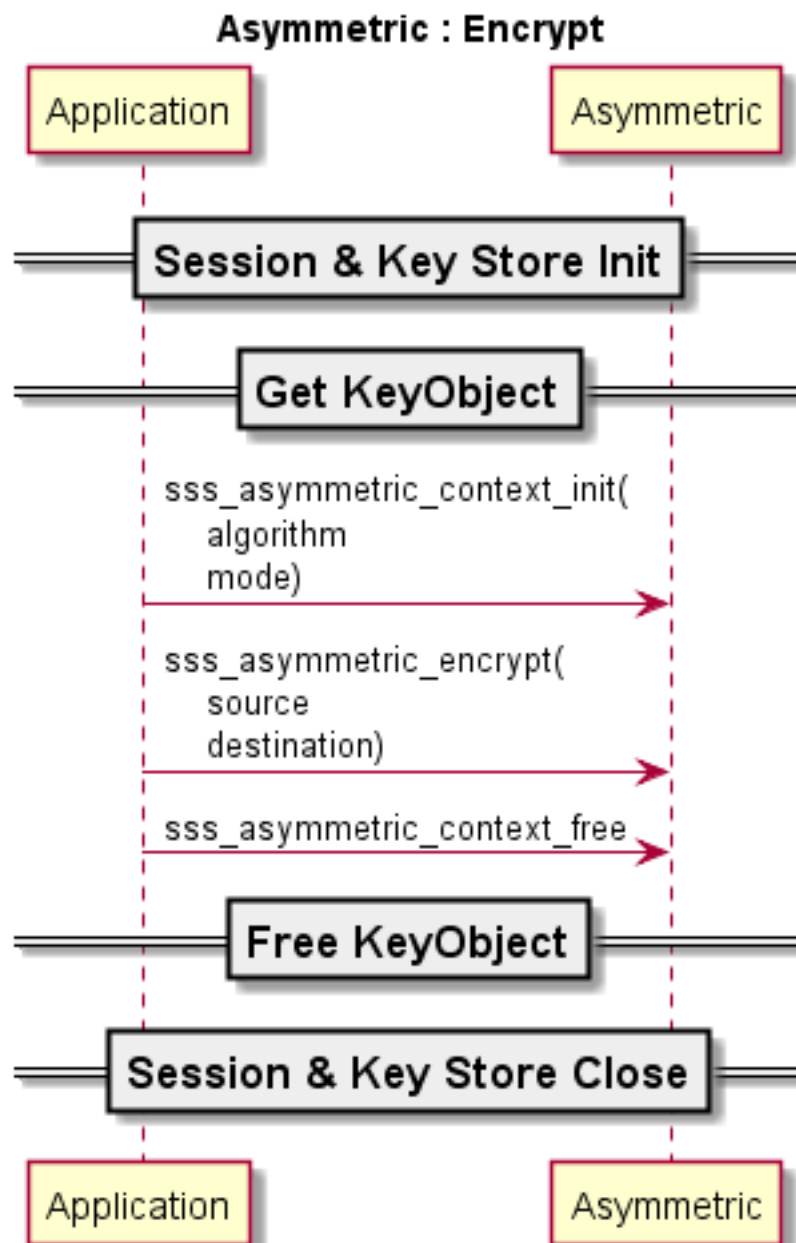
Verify

To perform sign verify operation , the sequence of APIs looks as under:



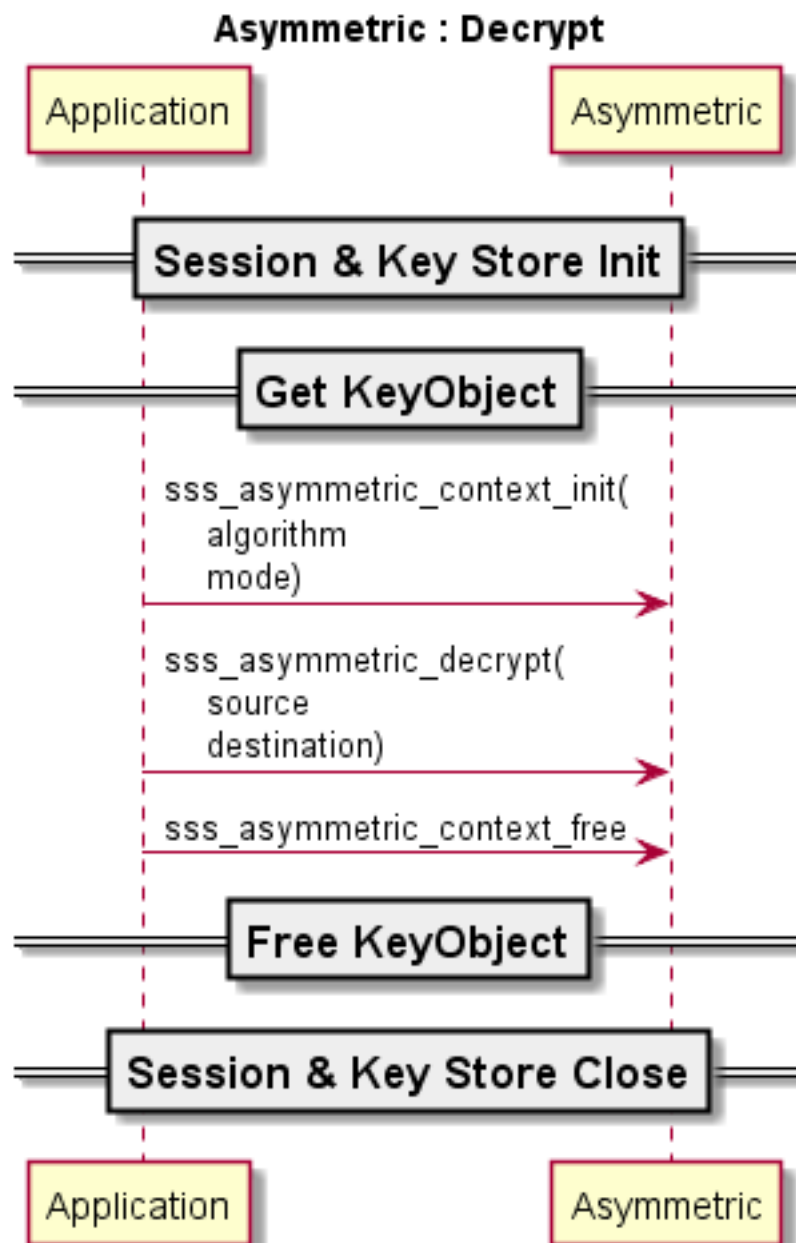
Encryption

To encrypt the data , the API sequence is as under:



Decryption

To Decrypt the encrypted data , the API sequence is as under:



Reference Example

Before we use any Cryptographic operations, we need relevant Keys to be declared.

Here is a reference snippet to *inject* a key into the Secure Domain. (If the key was already existing in the Key Store, these steps are not needed)

```
/* Pre-requisite for Signing Part*/
status = sss_key_object_init(&keyPair, &pCtx->ks);
ENSURE_OR_GO_CLEANUP(status == kStatus_SSS_Success);

status = sss_key_object_allocate_handle(&keyPair,
    MAKE_TEST_ID(__LINE__),
    kSSS_KeyPart_Pair,
    kSSS_CipherType_EC_NIST_P,
    sizeof(keyPairData),
    kKeyObject_Mode_Persistent);
ENSURE_OR_GO_CLEANUP(status == kStatus_SSS_Success);

status = sss_key_store_set_key(&pCtx->ks, &keyPair, keyPairData,
↪ sizeof(keyPairData), EC_KEY_BIT_LEN, NULL, 0);
ENSURE_OR_GO_CLEANUP(status == kStatus_SSS_Success);
```

Signing on a digest of length digestLen is performed as below.

```
status = sss_asymmetric_context_init(&ctx_asymm, &pCtx->session, &keyPair,
↪ kAlgorithm_SSS_SHA256, kMode_SSS_Sign);
ENSURE_OR_GO_CLEANUP(status == kStatus_SSS_Success);

signatureLen = sizeof(signature);
/* Do Signing */
LOG_I("Do Signing");
LOG_MAU8_I("digest", digest, digestLen);
status = sss_asymmetric_sign_digest(&ctx_asymm, digest, digestLen, signature, &
↪ signatureLen);
ENSURE_OR_GO_CLEANUP(status == kStatus_SSS_Success);
LOG_MAU8_I("signature", signature, signatureLen);
LOG_I("Signing Successful !!!");
sss_asymmetric_context_free(&ctx_asymm);
```

After the above operation, signature has the signature using the key object keyPair.

RSA Encryption algorithms supported

Supported rsa encryption / decryption algorithms - PKCS1_OAEP and PKCS1_V1_5.

```
kAlgorithm_SSS_RSAES_PKCS1_OAEP_SHA1    = SSS_ENUM_ALGORITHM(RSAES_PKCS1_OAEP,
↪ 0x01),
kAlgorithm_SSS_RSAES_PKCS1_OAEP_SHA224 = SSS_ENUM_ALGORITHM(RSAES_PKCS1_OAEP,
↪ 0x02),
kAlgorithm_SSS_RSAES_PKCS1_OAEP_SHA256 = SSS_ENUM_ALGORITHM(RSAES_PKCS1_OAEP,
↪ 0x03),
kAlgorithm_SSS_RSAES_PKCS1_OAEP_SHA384 = SSS_ENUM_ALGORITHM(RSAES_PKCS1_OAEP,
↪ 0x04),
kAlgorithm_SSS_RSAES_PKCS1_OAEP_SHA512 = SSS_ENUM_ALGORITHM(RSAES_PKCS1_OAEP,
↪ 0x05),
```

(continues on next page)

(continued from previous page)

```
kAlgorithm_SSS_RSAES_PKCS1_V1_5_SHA1 = SSS_ENUM_ALGORITHM(RSAES_PKCS1_V1_5, ↵
↵ 0x01),
kAlgorithm_SSS_RSAES_PKCS1_V1_5_SHA224 = SSS_ENUM_ALGORITHM(RSAES_PKCS1_V1_5, ↵
↵ 0x02),
kAlgorithm_SSS_RSAES_PKCS1_V1_5_SHA256 = SSS_ENUM_ALGORITHM(RSAES_PKCS1_V1_5, ↵
↵ 0x03),
kAlgorithm_SSS_RSAES_PKCS1_V1_5_SHA384 = SSS_ENUM_ALGORITHM(RSAES_PKCS1_V1_5, ↵
↵ 0x04),
kAlgorithm_SSS_RSAES_PKCS1_V1_5_SHA512 = SSS_ENUM_ALGORITHM(RSAES_PKCS1_V1_5, ↵
↵ 0x05),
```

RSA Signature algorithms supported

Supported rsa sign / verify algorithms - PKCS1_PSS_MGF1, PKCS1_V1_5 and No_Padding.

Hash algorithms supported for sign/verify - SHA1, SHA224, SHA256, SHA384, SHA512

```
kAlgorithm_SSS_RSASSA_PKCS1_V1_5_NO_HASH = SSS_ENUM_ALGORITHM(RSASSA_PKCS1_V1_
↵ 5, 0x01),
kAlgorithm_SSS_RSASSA_PKCS1_V1_5_SHA1 = SSS_ENUM_ALGORITHM(RSASSA_PKCS1_V1_
↵ 5, 0x02),
kAlgorithm_SSS_RSASSA_PKCS1_V1_5_SHA224 = SSS_ENUM_ALGORITHM(RSASSA_PKCS1_V1_
↵ 5, 0x03),
kAlgorithm_SSS_RSASSA_PKCS1_V1_5_SHA256 = SSS_ENUM_ALGORITHM(RSASSA_PKCS1_V1_
↵ 5, 0x04),
kAlgorithm_SSS_RSASSA_PKCS1_V1_5_SHA384 = SSS_ENUM_ALGORITHM(RSASSA_PKCS1_V1_
↵ 5, 0x05),
kAlgorithm_SSS_RSASSA_PKCS1_V1_5_SHA512 = SSS_ENUM_ALGORITHM(RSASSA_PKCS1_V1_
↵ 5, 0x06),
kAlgorithm_SSS_RSASSA_PKCS1_PSS_MGF1_SHA1 = SSS_ENUM_ALGORITHM(RSASSA_PKCS1_PSS_
↵ MGF1, 0x01),
kAlgorithm_SSS_RSASSA_PKCS1_PSS_MGF1_SHA224 = SSS_ENUM_ALGORITHM(RSASSA_PKCS1_PSS_
↵ MGF1, 0x02),
kAlgorithm_SSS_RSASSA_PKCS1_PSS_MGF1_SHA256 = SSS_ENUM_ALGORITHM(RSASSA_PKCS1_PSS_
↵ MGF1, 0x03),
kAlgorithm_SSS_RSASSA_PKCS1_PSS_MGF1_SHA384 = SSS_ENUM_ALGORITHM(RSASSA_PKCS1_PSS_
↵ MGF1, 0x04),
kAlgorithm_SSS_RSASSA_PKCS1_PSS_MGF1_SHA512 = SSS_ENUM_ALGORITHM(RSASSA_PKCS1_PSS_
↵ MGF1, 0x05),
```

```
kAlgorithm_SSS_RSASSA_NO_PADDING = SSS_ENUM_ALGORITHM(RSASSA_NO_PADDING, 0x01),
```

When using PKCS1_PSS_MGF1 padding, there are few limitations on hash algorithm with rsa key size as below,

RSA Bit Length	Valid Hash Algorithm
512	SHA1, SHA224
1024	SHA1, SHA224, SHA256, SHA384
1152	SHA1, SHA224, SHA256, SHA384, SHA512
2048	SHA1, SHA224, SHA256, SHA384, SHA512
3072	SHA1, SHA224, SHA256, SHA384, SHA512
4096	SHA1, SHA224, SHA256, SHA384, SHA512

ECC Signature algorithms supported

Supported hash values for ecc sign / verify - SHA1, SHA224, SHA256, SHA384, SHA512

```
kAlgorithm_SSS_ECDSA_SHA1    = SSS_ENUM_ALGORITHM(ECDSA, 0x01),
kAlgorithm_SSS_ECDSA_SHA224  = SSS_ENUM_ALGORITHM(ECDSA, 0x02),
kAlgorithm_SSS_ECDSA_SHA256  = SSS_ENUM_ALGORITHM(ECDSA, 0x03),
kAlgorithm_SSS_ECDSA_SHA384  = SSS_ENUM_ALGORITHM(ECDSA, 0x04),
kAlgorithm_SSS_ECDSA_SHA512  = SSS_ENUM_ALGORITHM(ECDSA, 0x05),
```

OR

```
kAlgorithm_SSS_SHA1          = SSS_ENUM_ALGORITHM(SHA, 0x01),
kAlgorithm_SSS_SHA224        = SSS_ENUM_ALGORITHM(SHA, 0x02),
kAlgorithm_SSS_SHA256        = SSS_ENUM_ALGORITHM(SHA, 0x03),
kAlgorithm_SSS_SHA384        = SSS_ENUM_ALGORITHM(SHA, 0x04),
kAlgorithm_SSS_SHA512        = SSS_ENUM_ALGORITHM(SHA, 0x05),
```

APIs

group **sss_crypto_asymmetric**

Asymmetric cryptographic operations like RSA / ECC / etc.

Functions

void **sss_asymmetric_context_free** (*sss_asymmetric_t* *context)

Asymmetric context release. The function frees asymmetric context.

Parameters

- context: Pointer to asymmetric context.

sss_status_t **sss_asymmetric_context_init** (*sss_asymmetric_t* *context, *sss_session_t* *session, *sss_object_t* *keyObject, *sss_algorithm_t* algorithm, *sss_mode_t* mode)

Asymmetric context init. The function initializes asymmetric context with initial values.

Return Status of the operation

Parameters

- context: Pointer to asymmetric crypto context.
- session: Associate SSS session with asymmetric context.
- keyObject: Associate SSS key object with asymmetric context.
- algorithm: One of the asymmetric algorithms defined by *sss_algorithm_t*.
- mode: One of the modes defined by *sss_mode_t*.

Return Value

- *kStatus_SSS_Success*: The operation has completed successfully.
- *kStatus_SSS_Fail*: The operation has failed.
- *kStatus_SSS_InvalidArgument*: One of the arguments is invalid for the function to execute.

sss_status_t **sss_asymmetric_decrypt** (*sss_asymmetric_t* *context, const uint8_t *srcData, size_t srcLen, uint8_t *destData, size_t *destLen)

Asymmetric decryption The function uses asymmetric algorithm to decrypt data. Private key portion of a key pair is used for decryption.

Return Status of the operation

Parameters

- context: Pointer to asymmetric context.
- srcData: Input buffer
- srcLen: Length of the input in bytes
- destData: Output buffer
- destLen: Length of the output in bytes

Return Value

- *kStatus_SSS_Success*: The operation has completed successfully.
- *kStatus_SSS_Fail*: The operation has failed.
- *kStatus_SSS_InvalidArgument*: One of the arguments is invalid for the function to execute.

sss_status_t **sss_asymmetric_encrypt** (*sss_asymmetric_t* *context, const uint8_t *srcData, size_t srcLen, uint8_t *destData, size_t *destLen)

Asymmetric encryption The function uses asymmetric algorithm to encrypt data. Public key portion of a key pair is used for encryption.

Return Status of the operation

Parameters

- context: Pointer to asymmetric context.
- srcData: Input buffer
- srcLen: Length of the input in bytes
- destData: Output buffer
- destLen: Length of the output in bytes

Return Value

- *kStatus_SSS_Success*: The operation has completed successfully.
- *kStatus_SSS_Fail*: The operation has failed.
- *kStatus_SSS_InvalidArgument*: One of the arguments is invalid for the function to execute.

sss_status_t **sss_asymmetric_sign_digest** (*sss_asymmetric_t* *context, uint8_t *digest, size_t digestLen, uint8_t *signature, size_t *signatureLen)

Asymmetric signature of a message digest The function signs a message digest.

Return Status of the operation

Parameters

- context: Pointer to asymmetric context.
- digest: Input buffer containing the input message digest

- `digestLen`: Length of the digest in bytes
- `signature`: Output buffer written with the signature of the digest
- `signatureLen`: Length of the signature in bytes

Return Value

- `kStatus_SSS_Success`: The operation has completed successfully.
- `kStatus_SSS_Fail`: The operation has failed.
- `kStatus_SSS_InvalidArgument`: One of the arguments is invalid for the function to execute.

`sss_status_t sss_asymmetric_verify_digest` (`sss_asymmetric_t *context`, `uint8_t *digest`, `size_t digestLen`, `uint8_t *signature`, `size_t signatureLen`)

Asymmetric verify of a message digest The function verifies a message digest.

Return Status of the operation

Parameters

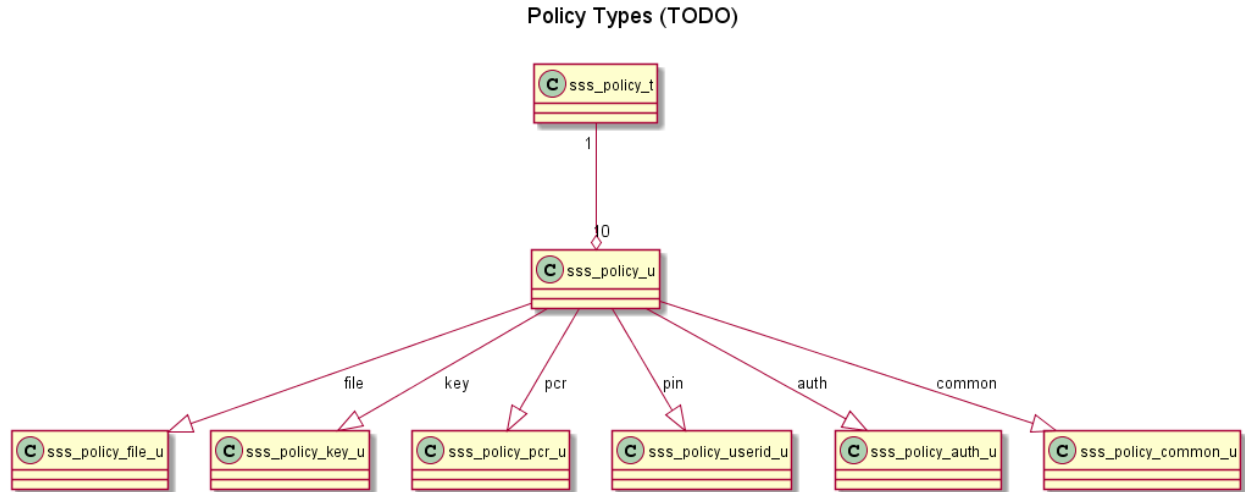
- `context`: Pointer to asymmetric context.
- `digest`: Input buffer containing the input message digest
- `digestLen`: Length of the digest in bytes
- `signature`: Input buffer containing the signature to verify
- `signatureLen`: Length of the signature in bytes

Return Value

- `kStatus_SSS_Success`: The operation has completed successfully.
- `kStatus_SSS_Fail`: The operation has failed.
- `kStatus_SSS_InvalidArgument`: One of the arguments is invalid for the function to execute.

3.3.6 Policies

Policies can be used to restrict & control the usage of session or objects.



Usage

Policy can be declared like below:

```

/* Policies for key */
const sss_policy_u key_withPol = {
    .type = KPolicy_Asym_Key,
    /*Authentication object based on SE05X_AUTH*/
    .auth_obj_id = EX_LOCAL_OBJ_AUTH_ID,
    .policy = {
        /*Asymmetric key policy*/
        .asymmkey = {
            /*Policy for sign*/
            .can_Sign = enable,
            /*Policy for verify*/
            .can_Verify = 1,
            /*Policy for encrypt*/
            .can_Encrypt = 1,
            /*Policy for decrypt*/
            .can_Decrypt = 1,
            /*Policy for Key Derivation*/
            .can_KD = 1,
            /*Policy for wrapped object*/
            .can_Wrap = 1,
            /*Policy to re-write object*/
            .can_Write = 1,
            /*Policy for reading object*/
            .can_Read = 1,
            /*Policy to use object for attestation*/
            .can_Attest = 1,
        }
    }
};

/* Common rules */
const sss_policy_u common = {
    .type = KPolicy_Common,
    /*Authentication object based on SE05X_AUTH*/
    .auth_obj_id = EX_LOCAL_OBJ_AUTH_ID,

```

(continues on next page)

(continued from previous page)

```

        .policy = {
        .common = {
        /*Secure Messaging*/
        .req_Sm = 0,
        /*Policy to Delete object*/
        .can_Delete = 1,
        /*Forbid all operations on object*/
        .forbid_All = 0,
        }
        }
    };

    /* create policy set */
    sss_policy_t policy_for_ec_key = {
        .nPolicies = 2,
        .policies = { &key_withPol, &common }
    };

```

To create an object with that policy, usage is as below:

```

status = sss_key_store_generate_key(
    &pCtx->ks,
    &object,
    ECC_KEY_BIT_LEN,
    &policy_for_ec_key);

```

APIs

group **sss_policy**

Policies to restrict and control sessions and objects.

Enums

enum **sss_policy_type_u**

Type of policy

Values:

KPolicy_None

No policy applied

KPolicy_Session

Policy related to session.

See [sss_policy_session_u](#)

KPolicy_Sym_Key

Policy related to key.

See [sss_policy_key_u](#)

KPolicy_Asym_Key

KPolicy_UserID

KPolicy_File

KPolicy_Counter

```

KPolicy_PCR
KPolicy_Common
KPolicy_Common_PCR_Value
struct sss_policy_asym_key_u
    #include <fsl_oss_policy.h> Policies applicable to Asymmetric KEY

```

Public Members

```

uint8_t can_Attest
    Allow to attest an object

uint8_t can_Decrypt
    Allow decryption

uint8_t can_Encrypt
    Allow encryption

uint8_t can_Gen
    Allow to (re)generate the object

uint8_t can_Import_Export
    Allow to imported or exported

uint8_t can_KA
    Allow key agreement

uint8_t can_KD
    Allow key derivation

uint8_t can_Read
    Allow to read the object

uint8_t can_Sign
    Allow signature generation

uint8_t can_Verify
    Allow signature verification

uint8_t can_Wrap
    Allow key wrapping

uint8_t can_Write
    Allow to write the object

uint8_t forbid_Derived_Output
    Forbid derived output

struct sss_policy_common_pcr_value_u
    #include <fsl_oss_policy.h> Common PCR Value Policies for all object types

```


Public Members

uint8_t **pcrExpectedValue**[32]
Expected value of the PCR

uint32_t **pcrObjId**
PCR object ID

struct sss_policy_common_u
#include <fsl_sss_policy.h> Common Policies for all object types

Public Members

uint8_t **can_Delete**
Allow to delete the object

uint8_t **forbid_All**
Forbid all operations

uint8_t **req_Sm**
Require having secure messaging enabled with encryption and integrity on the command

struct sss_policy_counter_u
#include <fsl_sss_policy.h> All policies related to secure object type Counter

Public Members

uint8_t **can_Read**
Allow to read the object

uint8_t **can_Write**
Allow to write the object

struct sss_policy_file_u
#include <fsl_sss_policy.h> All policies related to secure object type File

Public Members

uint8_t **can_Read**
Allow to read the object

uint8_t **can_Write**
Allow to write the object

struct sss_policy_pcr_u
#include <fsl_sss_policy.h> All policies related to secure object type PCR

Public Members

uint8_t can_Read
Allow to read the object

uint8_t can_Write
Allow to write the object

struct sss_policy_session_u
#include <fsl_sss_policy.h> Policy applicable to a session

Public Members

uint8_t allowRefresh
Whether this session can be refreshed without losing context. And also reset maxDurationOfSession_sec / maxOperationsInSession

uint8_t has_MaxDurationOfSession_sec
Whether maxOperationsInSession is set. This is to ensure '0 == maxDurationOfSession_sec' does not get set by middleware.

uint8_t has_MaxOperationsInSession
Whether maxOperationsInSession is set. This is to ensure '0 == maxOperationsInSession' does not get set by middleware.

uint16_t maxDurationOfSession_sec
Session can be used for this much time, in seconds

uint16_t maxOperationsInSession
Number of operations permitted in a session

struct sss_policy_sym_key_u
#include <fsl_sss_policy.h> Policies applicable to Symmetric KEY

Public Members

uint8_t can_Decrypt
Allow decryption

uint8_t can_Desfire_Auth
Allow to perform DESFire authentication

uint8_t can_Desfire_Dump
Allow to dump DESFire session keys

uint8_t can_Encrypt
Allow encryption

uint8_t can_Gen
Allow to (re)generate the object

uint8_t can_Import_Export
Allow to imported or exported

uint8_t can_KD
Allow key derivation

uint8_t can_Sign
Allow signature generation

```

uint8_t can_Verify
    Allow signature verification

uint8_t can_Wrap
    Allow key wrapping

uint8_t can_Write
    Allow to write the object

uint8_t forbid_Derived_Output
    Forbid derived output

struct sss_policy_t
    #include <fsl_sss_policy.h> An array of policies sss_policy_u

```

Public Members

```

size_t nPolicies
    Number of policies

const sss_policy_u *policies[(10)]
    Array of unique policies, this needs to be allocated based nPolicies

struct sss_policy_u
    #include <fsl_sss_policy.h> Unique/individual policy. For any operation, you need array of sss_policy_u.

```

Public Members

```

sss_policy_asym_key_u asymmkey

uint32_t auth_obj_id
    Auth ID for each Object Policy, invalid for session policy type == KPolicy_Session

sss_policy_common_u common

sss_policy_common_pcr_value_u common_pcr_value

sss_policy_counter_u counter

sss_policy_file_u file

sss_policy_pcr_u pcr

sss_policy_userid_u pin

union sss_policy_u::[anonymous] policy
    Union of applicable policies based on the type of object

sss_policy_session_u session

sss_policy_sym_key_u symmkey

sss_policy_type_u type
    Secure Object Type

struct sss_policy_userid_u
    #include <fsl_sss_policy.h> All policies related to secure object type UserID

```

Public Members

`uint8_t can_Write`

Allow to write the object

3.3.7 Example Boot-Up

The Examples and use cases based on SSS APIs are them selves (*more or less*) Cryptosystem agnostic, how ever the platforms where they run and how they run would be very specific.

e.g. While running from PC with a Secure Element, you may need to choose and connect to specific COM Port / Socket. On the other hand, when running from different embedded platforms like FREEDOM K64F, iMX RT 1050, etc., few board specific steps are needed.

To simplify examples, them selves, the files in `sss/ex/inc` and `sss/ex/src` try to isolate such details.

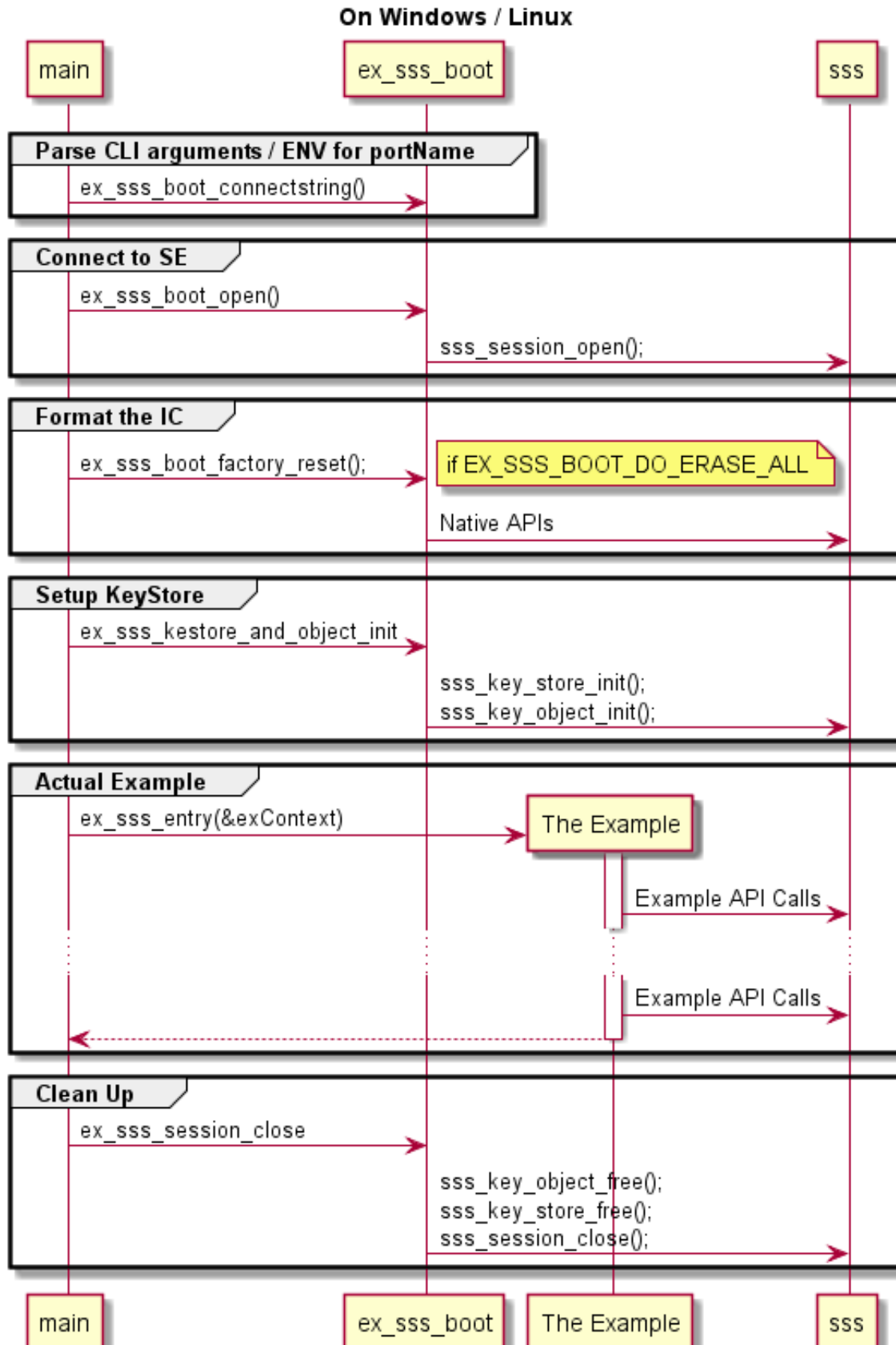
Some of the scenarios of boot up are:

Bootting from Windows / Linux

In such a system, many decisions are taken at run time. e.g. COM Port for interface to the secure element, etc.

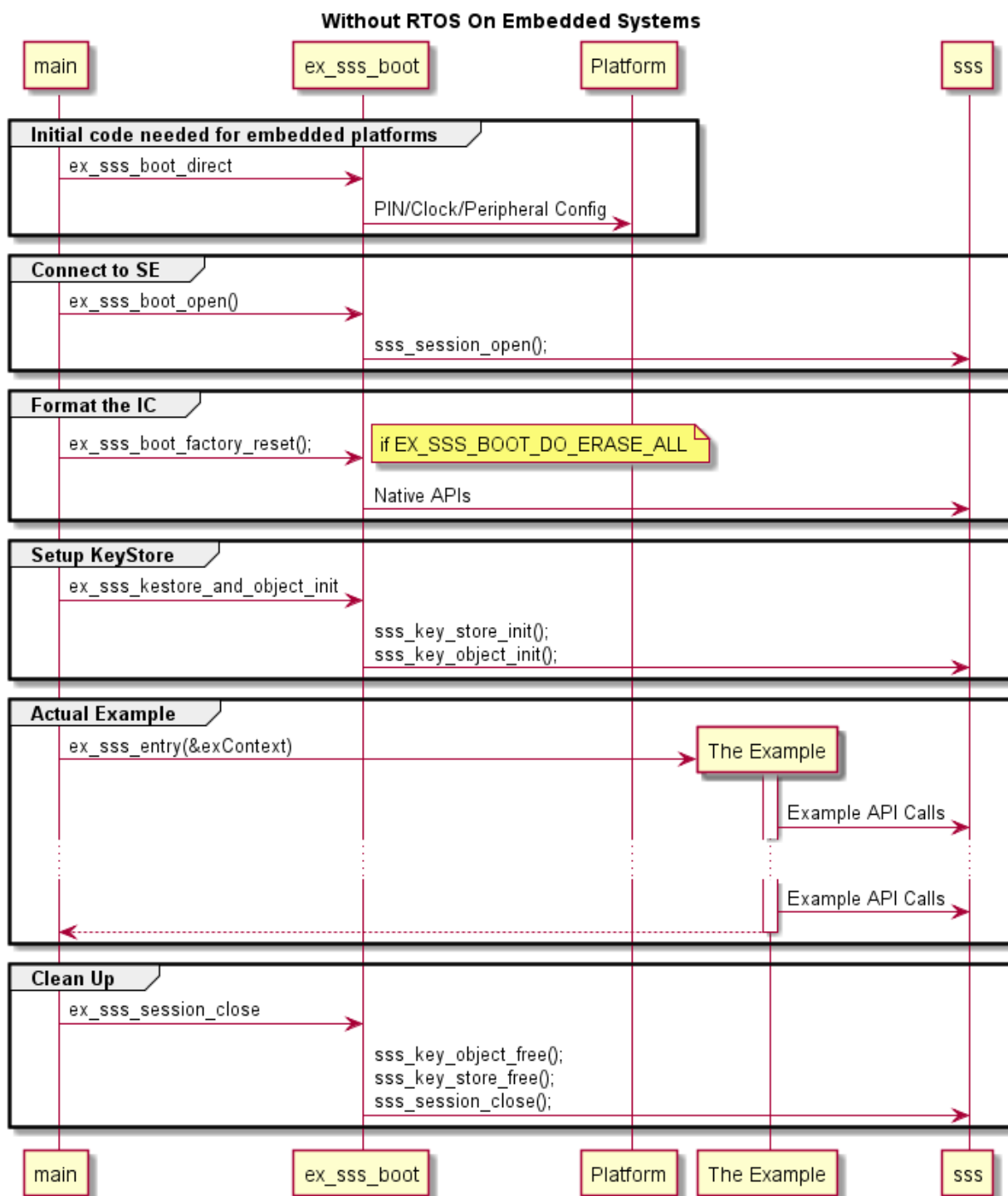
In such a system, examples also have access to command line arguments and environment variables.

Such a setup is mostly for testing and early prototyping.



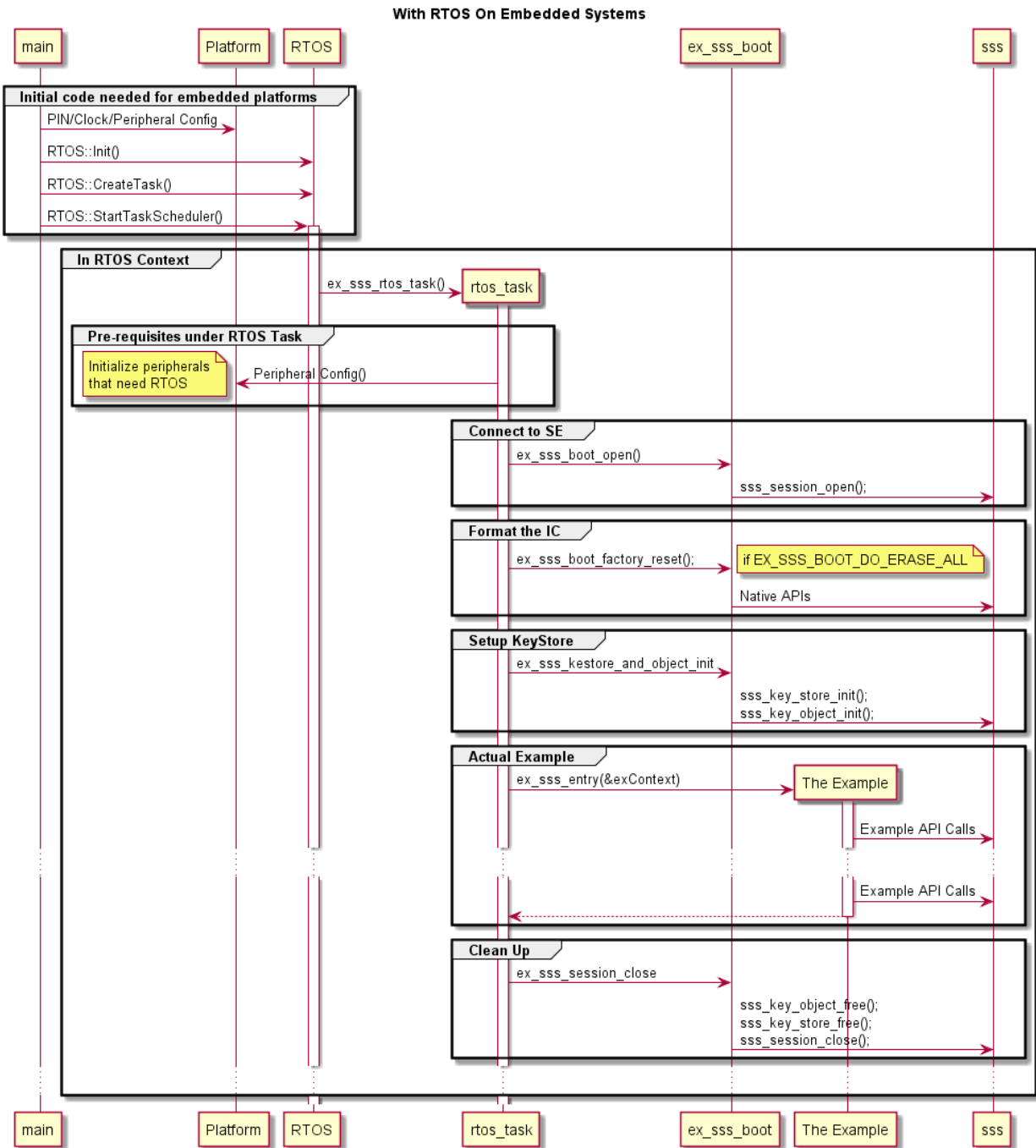
Booting from an embedded system, without any RTOS

In such a system, the example is pre-compiled for specific platform/combination. There are very less decisions to be taken at run time, and most decisions are pre-selected during build/compile time.



Booting from an embedded system, with RTOS

In such a system, the example uses RTOS. And the example itself is to be run from an RTOS Thread context.



3.3.8 SSS api key format (asymmetric keys)

NIST-P, NIST-K and BRAINPOOL ECC Keys

When passing the key pair / public key, the key should be passed DER encoded using pkcs8 or traditional openssl format. When passing the private key alone, do not include the key header.

When retrieving the key pair as data argument from the `sss_key_store_get` api, the full key pair cannot be retrieved. Instead the public key value is returned in ANSI X9.62 uncompressed format.

Note: Keys, signature and shared secret key generated all follow the big endian convention.

EDWARD and MONTGOMERY ECC keys

When passing the key pair / public key, the key should be passed DER encoded using the format specified in RFC 8410. When passing the private key alone, do not include the key header.

When retrieving the key pair as data argument from the `sss_key_store_get` api, the full key pair cannot be retrieved. Instead the public key value is returned in ANSI X9.62 uncompressed format.

Note: Keys, signature and shared secret key generated all follow the little endian convention.

A set of examples of a X25519 keypair encoded according to RFC 8410:

```
$ dumpasn1 X25519_keypair.der
0  81: SEQUENCE {
2   1:  INTEGER 1
5   5:  SEQUENCE {
7   3:    OBJECT IDENTIFIER curveX25519 (1 3 101 110)
:    }
12  34:  OCTET STRING, encapsulates {
14  32:    OCTET STRING
:      58 5D B1 E3 50 0B 71 24 F6 B1 E1 41 83 54 93 12
:      F4 4B 0C A3 44 F7 52 A1 8A 12 2F E7 DA D9 CE 52
:    }
48  33:  [1]
:      00 A2 8E 04 FF 1C DC 1C 3D 60 91 0F BC 98 EF 01
:      BF 9F 0F 69 C0 B7 EF 70 61 35 34 62 F3 06 28 C7
:      29
:    }

$ dumpasn1 X25519_priv.pem
0  46: SEQUENCE {
2   1:  INTEGER 0
5   5:  SEQUENCE {
7   3:    OBJECT IDENTIFIER curveX25519 (1 3 101 110)
:    }
12  34:  OCTET STRING, encapsulates {
14  32:    OCTET STRING
:      58 5D B1 E3 50 0B 71 24 F6 B1 E1 41 83 54 93 12
:      F4 4B 0C A3 44 F7 52 A1 8A 12 2F E7 DA D9 CE 52
:    }
:  }
```

(continues on next page)

(continued from previous page)

```
$ dumpasn1 X25519_pub.pem
0  42: SEQUENCE {
2   5:  SEQUENCE {
4    3:  OBJECT IDENTIFIER curveX25519 (1 3 101 110)
      :  }
9   33:  BIT STRING
      :  A2 8E 04 FF 1C DC 1C 3D 60 91 0F BC 98 EF 01 BF
      :  9F 0F 69 C0 B7 EF 70 61 35 34 62 F3 06 28 C7 29
      :  }
```

BN Curve ECC keys

When passing key pair, private key or public key do not include key header. When retrieving the key from the `sss_key_store_get` api, the public key value is returned without any header.

RSA keys

When passing the key pair / public key, the key should be passed DER encoded using pkcs8 or traditional openssl format. When passing the private key alone, do not include the key header.

When retrieving the key pair as data argument from the `sss_key_store_get` API, the full key pair cannot be retrieved. Instead the public key value is returned in ANSI X9.62 uncompressed format.

See also [Section 5.2 SSS API Examples](#)

3.4 Parameter Check & Conventions

3.4.1 Parameter Convention

APIs for which a buffer is input. e.g.:

```
smStatus_t Se05x_API_VerifySessionUserID(
    pSe05xSession_t session_ctx,
    const uint8_t *userId,
    size_t userIdLen);
```

In the above case `userId` is a buffer input. It is assumed that the length as set in `userIdLen` is same as that pointed to by `userId`. This parameter is used as is and any mistake by the calling API will have unpredictable errors.

APIs for which a buffer is input. e.g.:

```
smStatus_t Se05x_API_ReadObject(
    pSe05xSession_t session_ctx,
    uint32_t objectID,
    uint16_t offset,
    uint16_t length,
    uint8_t *data,
    size_t *pdataLen);
```

In the above case `data` is a buffer output and `pdataLen` is both input and output. It is assumed that the length as set in `pdataLen` is set to the maximum as available to the pointer pointed by `data`. This parameter is used as is and any mistake by the calling API will have unpredictable errors.

PCSC/CCID Interface and 64 Byte packet

See the note “USB 64 byte boundary” at `hostLib\\hostLib\\libCommon\\smCom\\smComPCSC.c`

3.4.2 Helper Macros

Helper macros are available as a part of the stack to capture warnings if some parameters are not as expected.

During debug builds, it is recommended to enable the logging to capture mistakes during integration.

During Retail/Release builds, they may be kept silent.

3.4.3 Apis

group **param_check**
Parameter Checks.

`nxEnsure.h`: Helper parameter assertion check macros.

Pre Condition: The source file must have included `nxLog` header file.

Project: SecureIoTMW

Defines

ENSURE_OR_BREAK (CONDITION)

If condition fails, break.

Sample Usage:

```
int SomeAPI()
{
    ...

    do {
        status = Operation1();
        ENSURE_OR_BREAK(0 == status);

        status = Operation2();
        ENSURE_OR_BREAK(0 == status);

        ...
    } while(0);

    return status;
}
```

ENSURE_OR_EXIT_WITH_STATUS_ON_ERROR (CONDITION, STATUS, RETURN_VALUE)

If condition fails, goto quit with return value status updated.

```
int SomeAPI()
{
    int status = 0;
    ...
```

(continues on next page)

(continued from previous page)

```

    value = Operation1();
    ENSURE_OR_QUIT_WITH_STATUS_ON_ERROR(0 == value, status, ERR_FAIL);

    value = Operation2();
    ENSURE_OR_QUIT_WITH_STATUS_ON_ERROR(0 == value, status, ERR_NOT_ENOUGH_
↪SPACE);

    ...
quit:
    return status;
}

```

Warning This macro introduces system of multiple returns from a function which is not easy to debug/trace through and hence not recommended.

ENSURE_OR_GO_CLEANUP (CONDITION)

If condition fails, goto :cleanup label

```

{
    ...

    status = Operation1();
    ENSURE_OR_GO_CLEANUP(0 == status);

    status = Operation2();
    ENSURE_OR_GO_CLEANUP(0 == status);

    ...

cleanup:
    return status;
}

```

ENSURE_OR_GO_EXIT (CONDITION)

If condition fails, goto :exit label

```

{
    ...

    status = Operation1();
    ENSURE_OR_GO_EXIT(0 == status);

    status = Operation2();
    ENSURE_OR_GO_EXIT(0 == status);

    ...

exit:
    return status;
}

```

ENSURE_OR_RETURN (CONDITION)

If condition fails, return

```

void SomeAPI()
{

```

(continues on next page)

(continued from previous page)

```
...

status = Operation1();
ENSURE_OR_RETURN(0 == status);

status = Operation2();
ENSURE_OR_RETURN(0 == status);

...

return;
}
```

Warning This macro introduces system of multiple returns from a function which is not easy to debug/trace through and hence not recommended.

ENSURE_OR_RETURN_ON_ERROR (CONDITION, RETURN_VALUE)

If condition fails, return

```
int SomeAPI()
{
    ...

    status = Operation1();
    ENSURE_OR_RETURN_ON_ERROR(0 == status, ERR_FAIL);

    status = Operation2();
    ENSURE_OR_RETURN_ON_ERROR(0 == status, ERR_NOT_ENOUGH_SPACE);

    ...

    return 0;
}
```

Warning This macro introduces system of multiple returns from a function which is not easy to debug/trace through and hence not recommended.

NX_ENSURE_DO_LOG_MESSAGE

Build time over-ride if we want to enable/disable Warning Prints

During debug builds, it makes sense to print them, During retail builds, such loggings would be of any use and remove and reduce code size.

NX_ENSURE_MESSAGE (strCONDITION)

Warning print of the parameter strCONDITION

Warning NX_ENSURE_MESSAGE is an internal message/API to this file. Do not use directly.

NX_ENSURE_MESSAGE (strCONDITION)

Warning print of the parameter strCONDITION

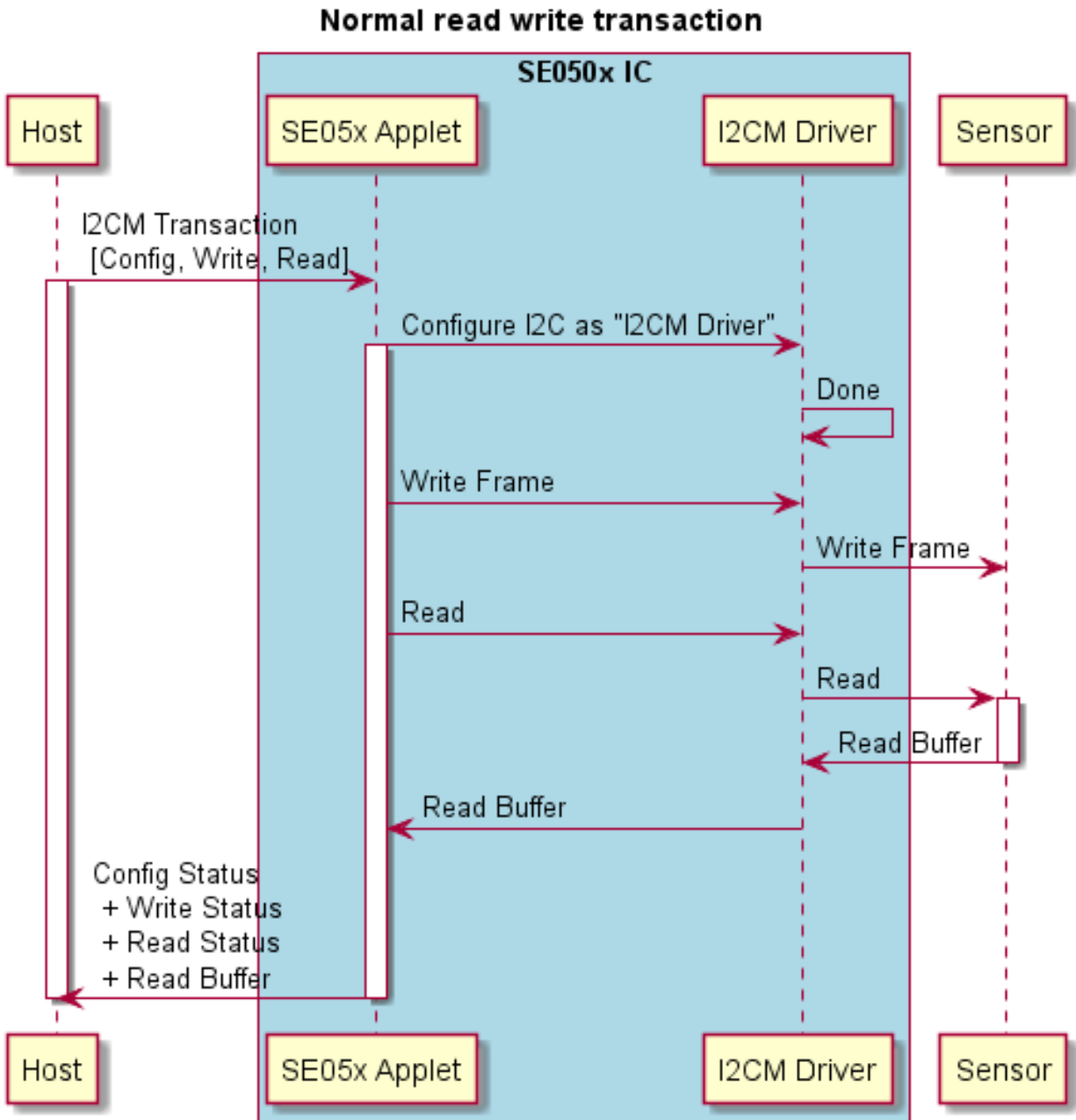
Warning NX_ENSURE_MESSAGE is an internal message/API to this file. Do not use directly.

3.5 I2CM / Secure Sensor

For an example regarding this, see [Section 5.20 I2C Master Example](#)

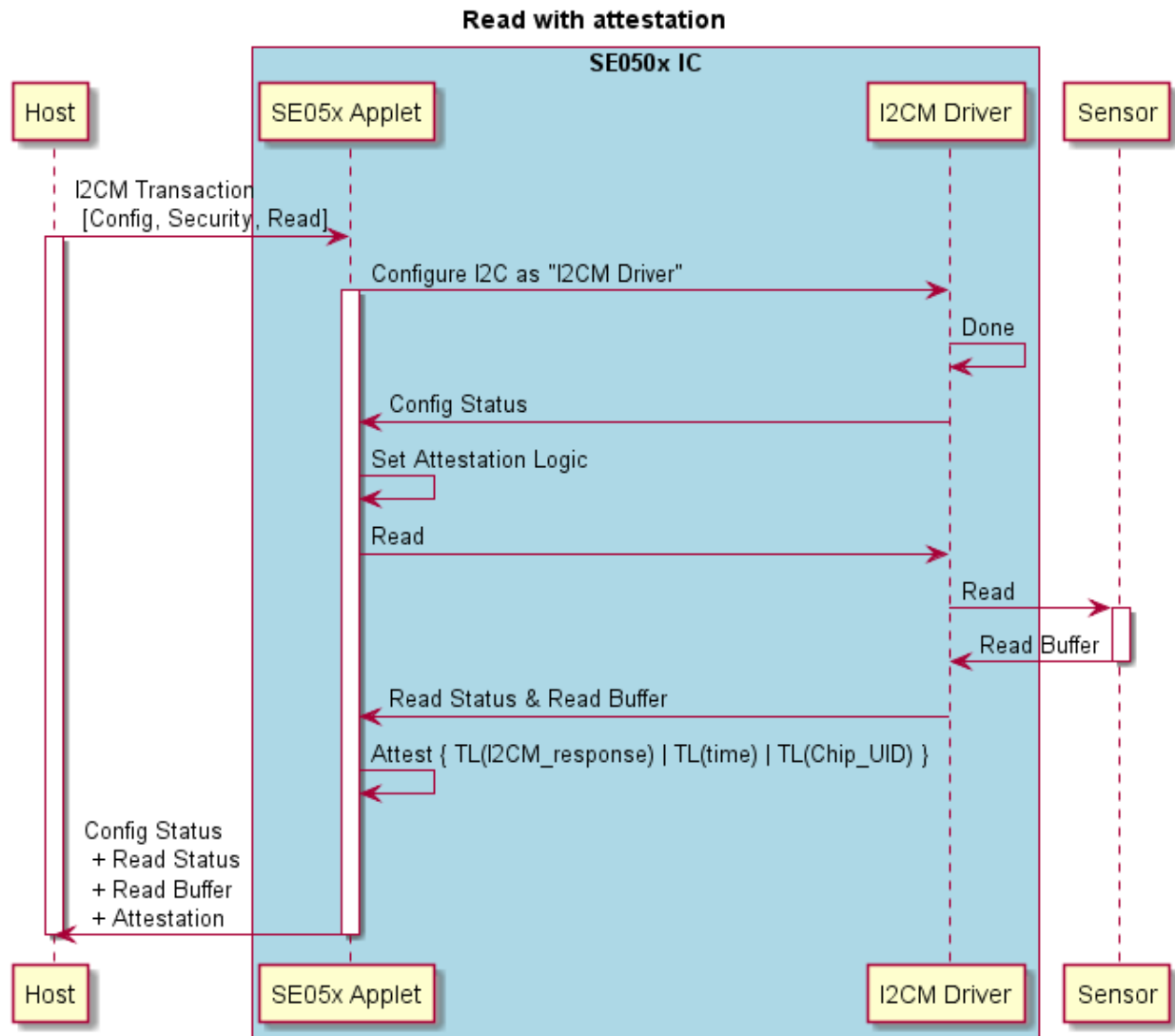
3.5.1 Normal Read/Write

The sequence to read from I2CM Sensor is as below:



3.5.2 Attested Read

The sequence to read with attestation from I2CM Sensor is as below:



3.5.3 Transaction

A sample I2CM Transaction can be performed as below:

Code:

```

FujAddr[0] = 0x03;
FujAddr[1] = 0x00;
TLV[0].type = kSE05x_I2CM_Configure;
TLV[0].cmd.cfg.I2C_addr = FUJITSU_I2C_ADDR;
TLV[0].cmd.cfg.I2C_baudRate = kSE05x_I2CM_Baud_Rate_100Khz;

TLV[1].type = kSE05x_I2CM_Write;
TLV[1].cmd.w.writeLength = sizeof(FujAddr);
  
```

(continues on next page)

(continued from previous page)

```

TLV[1].cmd.w.writebuf = FujAddr;

TLV[2].type = kSE05x_I2CM_Read;
TLV[2].cmd.rd.readLength = I2C_MAX_DATA;
TLV[2].cmd.rd.rdBuf = pFujMemRead;

status = Se05x_i2c_master_txn(&gtCtx.session, &TLV[0], 3);

```

3.5.4 Read with Attestation

A sample I2CM read with Attestation can be performed as below:

Code:

```

TLV[0].type = kSE05x_I2CM_Configure;
TLV[0].cmd.cfg.I2C_addr = FUJITSU_I2C_ADDR;
TLV[0].cmd.cfg.I2C_baudRate = kSE05x_I2CM_Baud_Rate_400Khz;

TLV[1].type = kSE05x_I2CM_Write;
TLV[1].cmd.w.writeLength = sizeof(FujAddr);
TLV[1].cmd.w.writebuf = FujAddr;

TLV[2].type = kSE05x_I2CM_Read;
TLV[2].cmd.rd.readLength = I2C_MAX_DATA;
TLV[2].cmd.rd.rdBuf = pFujMemRead;

signatureLen = MAX_SIGNATURE_LEN;
status = Se05x_i2c_master_attst_txn(&gtCtx.session,
    &gtCtx.key,
    &TLV[0],
    random,
    sizeof(random),
    attest_algo,
    &timeStamp,
    &timeStampLen,
    outrandom,
    &outrandomLen,
    chipId,
    &chipIdLen,
    signature,
    &signatureLen,
    3);

```

3.5.5 I2C Master APIs

group **se050_i2cm**

I2C Master APIs in SE050 for secure sensor.

Enums

enum SE05x_I2CM_Baud_Rate_t

Configuration for I2CM

Values:

kSE05x_I2CM_Baud_Rate_100Khz = 0

kSE05x_I2CM_Baud_Rate_400Khz

enum SE05x_I2CM_securityReq_t

Additional operation on data read by I2C

Values:

kSE05x_Security_None = 0

kSE05x_Sign_Request

kSE05x_Sign_Enc_Request

enum SE05x_I2CM_status_t

Status of I2CM Transaction

Values:

kSE05x_I2CM_Success = 0x5A

kSE05x_I2CM_I2C_Nack_Fail = 0x01

kSE05x_I2CM_I2C_Write_Error = 0x02

kSE05x_I2CM_I2C_Read_Error = 0x03

kSE05x_I2CM_I2C_Time_Out_Error = 0x05

kSE05x_I2CM_Invalid_Tag = 0x11

kSE05x_I2CM_Invalid_Length = 0x12

kSE05x_I2CM_Invalid_Length_Encode = 0x13

kSE05x_I2CM_I2C_Config = 0x21

enum SE05x_I2CM_TAG_t

I2C Master micro operation.

Values:

kSE05x_TAG_I2CM_Config = 0x01

kSE05x_TAG_I2CM_Write = 0x03

kSE05x_TAG_I2CM_Read = 0x04

enum SE05x_I2CM_TLV_type_t

Types of entries in an I2CM Transaction

Values:

kSE05x_I2CM_None = 0

Do nothing

kSE05x_I2CM_Configure

Configure the address, baudrate

kSE05x_I2CM_Write = 3

Write to I2C Slave

kSE05x_I2CM_Read

Read from I2C Slave

kSE05x_I2CM_StructuralIssue = 0xFF

Response from SE05x that there is something wrong

Functions

```
smStatus_t Se05x_i2c_master_attst_txn(sss_session_t *sess, sss_object_t *keyObject,
                                       SE05x_I2CM_cmd_t *p, uint8_t *random_attst,
                                       size_t random_attstLen, SE05x_AttestationAlgo_t
                                       attst_algo, SE05x_TimeStamp_t *ptimeStamp, size_t
                                       *ptimeStampLen, uint8_t *freshness, size_t *pfresh-
                                       nessLen, uint8_t *chipId, size_t *pchipIdLen, uint8_t
                                       *signature, size_t *psignatureLen, uint8_t noOfTags)
```

Se05x_i2c_master_attst_txn.

I2CM Read With Attestation

Pre p describes I2C master commands.

Post p contains execution state of I2C master commands, the I2C master commands can be overwritten to report on execution failure.

Parameters

- [in] sess: session identifier
- [in] keyObject: Keyobject which contains 4 byte attestaion KeyId
- [inout] p: Array of structure type capturing a sequence of i2c master cmd/rsp transactions.
- [in] random_attst: 16-byte freshness random
- [in] random_attstLen: length of freshness random
- [in] attst_algo: 1 byte attestationAlgo
- [out] ptimeStamp: timestamp
- [out] timeStampLen: Length for timestamp
- [out] freshness: freshness (random)
- [out] pfreshnessLen: Length for freshness
- [out] chipId: unique chip Id
- [out] pchipIdLen: Length for chipId
- [out] signature: signature
- [out] psignatureLen: Length for signature
- [in] noOfTags: Amount of structures contained in p

```
smStatus_t Se05x_i2c_master_txn(sss_session_t *sess, SE05x_I2CM_cmd_t *cmds, uint8_t
                                cmdLen)
```

Se05x_i2c_master_txn.

I2CM Transaction

Pre p describes I2C master commands.

Post p contains execution state of I2C master commands, the I2C master commands can be overwritten to report on execution failure.

Parameters

- [in] sess: session identifier
- [inout] cmds: Array of structure type capturing a sequence of i2c master cmd/rsp transactions.
- [in] cmdLen: Amount of structures contained in cmds

struct SE05x_I2CM_cmd_t

#include <fsl_sss_se05x_types.h> Individual entry in array of TLV commands, with type

Se05x_i2c_master_txn would expect an array of these.

Public Members

SE05x_I2CM_INS_type_t cmd

Individual entry in array of TLV commands.

SE05x_I2CM_TLV_type_t type

struct SE05x_I2CM_configData_t

#include <fsl_sss_se05x_types.h> Data Configuration for I2CM

Public Members

uint8_t I2C_addr

7 Bit address of I2C slave

SE05x_I2CM_Baud_Rate_t I2C_baudRate

What baud rate

SE05x_I2CM_status_t status

return status of the config operation

union SE05x_I2CM_INS_type_t

#include <fsl_sss_se05x_types.h> Individual entry in array of TLV commands.

Public Members

SE05x_I2CM_configData_t cfg

Data Configuration for I2CM

SE05x_I2CM_structuralIssue_t issue

Used to report error response, not for outgoing command

SE05x_I2CM_readData_t rd

Read to I2CM from I2C Slave

SE05x_I2CM_securityData_t sec

Security Configuration for I2CM.

SE05x_I2CM_writeData_t w

Write From I2CM to I2C Slave.

```
struct SE05x_I2CM_readData_t
    #include <fsl_sss_se05x_types.h> Read to I2CM from I2C Slave
```

Public Members

uint8_t ***rdBuf**
Output. rdBuf will point to Host buffer.

SE05x_I2CM_status_t **rdStatus**
[Out] status of the operation

uint16_t **readLength**
How many bytes to read

```
struct SE05x_I2CM_securityData_t
    #include <fsl_sss_se05x_types.h> Security Configuration for I2CM.
```

Public Members

uint32_t **keyObject**
object used for the operation

SE05x_I2CM_securityReq_t **operation**
Additional operation on data read by I2C

```
struct SE05x_I2CM_structuralIssue_t
    #include <fsl_sss_se05x_types.h> Used to report error response, not for outgoing command
```

Public Members

SE05x_I2CM_status_t **issueStatus**
[Out] In case there is any structural issue

```
struct SE05x_I2CM_writeData_t
    #include <fsl_sss_se05x_types.h> Write From I2CM to I2C Slave.
```

Public Members

uint8_t ***writebuf**
Buffer to be written

uint8_t **writeLength**
How many bytes to write

SE05x_I2CM_status_t **wrStatus**
[Out] status of the operation

3.6 Logging

In order to efficiently debug and diagnose the Plug & Trust Middleware and its supported use-cases and examples, it supports logging. The logging Framework is written in such a way that embedded platforms are kept in mind and at priority. The choice of logging is done at compile time and not at run time like other high-level languages.

3.6.1 Logging level

The logging is divided into following levels:

Error Some kind of unrecoverable or unexpected error has happened and the behaviour from this point on is most likely to be out of specifications/expectations. The suffix for this in *Logging APIs* is **E**.

Warn Whatever happened is unexpected but not fatal, the severity of the warning requires the judgement of the user. The suffix for this in *Logging APIs* is **W**.

Info This is information for the user. The suffix for this in *Logging APIs* is **I**.

Debug These are verbose low level diagnostic debug messages and not to be used/enabled in normal circumstances. The suffix for this in *Logging APIs* is **D**.

3.6.2 Adding log messages into the source code

- 1) Add one of *Logging Header Files* to the C source code.

Warning: Only for C source code. Never add the logging files to header files, only add them to C source code.

- 2) Call applicable *Logging APIs* at respective *Logging level*
- 3) Re-compile and re-run the software.

3.6.3 Logging APIs

The following are the loggings APIs to be called from the source code.

LOG_I(...) Log any value. C language format specifiers and values can be used if needed.

LOG_X8_I(VALUE) Log a HEX number 8 bits wide

LOG_U8_I(VALUE) Log an unsigned decimal number 8 bits wide

LOG_X16_I(VALUE) Log a HEX number 16 bits wide

LOG_U16_I(VALUE) Log an unsigned decimal number 16 bits wide

LOG_X32_I(VALUE) Log a HEX number 32 bits wide

LOG_U32_I(VALUE) Log an unsigned number 32 bits wide

LOG_AU8_I(ARRAY, LEN) Log an array of 8bits of length LEN

LOG_MAU8_I(MESSAGE, ARRAY, LEN) Same as LOG_AU8_I, but use MESSAGE.

- The logging APIs effectively use preprocessor directives like # and ## and therefore, if the variable names are verbose enough, API calls like LOG_X16_I(statusOfDeleteKey) are enough to log response with relevant information, and efficient for the developer to add a logging instruction.

- The suffix `_I` can be replaced with `_E`, `_W` or `_D` based on *Logging level*. The convention of logging remains the same.
- Use HEX Values to log enums and other hexadecimal numbers. For items that are not treated as HEX (e.g. length), use the decimal logging APIs.

For example, the APIs can be called as below.

```
retStatus = DoAPDUTxRx_Case3( /* ... */
    rspbuf,
    &rspbufLen);
LOG_X16_D(retStatus);
LOG_AU8_D(rspbuf, rspbufLen);
```

If needed, the same can be made more verbose as below.

```
LOG_D("Sending FOO Command");
retStatus = DoAPDUTxRx_Case3( /* ... */
    rspbuf,
    &rspbufLen);

LOG_D("FOO retStatus=0x04X", retStatus);
LOG_MAU8_D("FOO Command", rspbuf, rspbufLen);
```

Logging - Information

Code:

```
uint32_t xu32val=0x12341234u;
uint8_t xu8val=0x44;

LOG_I("Values are xu32val=0x%08X xu8val=0x%02X", xu32val, xu8val);
```

Output:

```
App:INFO :Values are xu32val=0x12341234 xu8val=0x44
```

Logging - Variable Names

Code:

```
uint32_t xu32val=0x12341234u;
uint8_t xu8val=0x44;
unsigned int some_int_value = 783;
unsigned char some_byte_value = 96;

LOG_I("Values are:");
LOG_X8_I(xu8val);
LOG_U8_I(some_byte_value);
LOG_X16_I(xu8val);
LOG_U16_I(some_byte_value);
LOG_X32_I(xu32val);
LOG_U32_I(some_int_value);
```

(continues on next page)

(continued from previous page)

```
/* Logging that will be mis-intepreted */
LOG_X16_I(some_byte_value);
```

Output:

```
App:INFO :Values are:
App:INFO :xu8val=0x44
App:INFO :some_byte_value=96
App:INFO :xu8val=0x0044
App:INFO :some_byte_value=96
App:INFO :xu32val=0x12341234
App:INFO :some_int_value=783
App:INFO :some_byte_value=0x0060
```

Logging - Arrays

Code:

```
const uint8_t some_array[] = {
    0x5A, 0x5B, 0x5C, 0x5D,
    0x5E, 0x5F, 0x60, 0x61,
    0x62, 0x63, 0x64, 0x65,
    0x66, 0x67, 0x68, 0x69,
    0x6A, 0x6B, 0x6C, 0x6D};
const uint8_t buffer[] = {
    0x2A, 0x2B, 0x2C, 0x2D,
    0x2E, 0x2F, 0x30, 0x31,
    0x32, 0x33, 0x34, 0x35,
    0x36, 0x37, 0x38, 0x39,
    0x3A, 0x3B, 0x3C, 0x3D,
    0x3E, 0x3F, 0x40, 0x41,
    0x42, 0x43, 0x44, 0x45};
LOG_AU8_I(some_array, ARRAY_SIZE(some_array));
LOG_MAU8_I("meaningful name", buffer, ARRAY_SIZE(buffer));
```

Output:

```
App:INFO :some_array (Len=20)
 5A 5B 5C 5D    5E 5F 60 61    62 63 64 65    66 67 68 69
 6A 6B 6C 6D
App:INFO :meaningful name (Len=28)
 2A 2B 2C 2D    2E 2F 30 31    32 33 34 35    36 37 38 39
 3A 3B 3C 3D    3E 3F 40 41    42 43 44 45
```

Logging - Levels

Code:

```
uint32_t xu32val=0x12341234u;
LOG_X32_D(xu32val);
LOG_X32_I(xu32val);
LOG_X32_W(xu32val);
LOG_X32_E(xu32val);
```

Output:

```
App:INFO :xu32val=0x12341234
App:WARN :xu32val=0x12341234
App:ERROR:xu32val=0x12341234
```

3.6.4 Logging Header Files

Some of the header files for logging are as under.

- nxLog_UseCases.h** High level use cases.
- nxLog_App.h** Applications and tools.
- nxLog_VCOM.h** Logging specifically for VCOM Layer.
- nxLog_sss.h** Logging specifically for SSS Layer.
- nxLog_hostLib.h** Logging specifically for Host Library Layer.
- nxLog_smCom.h** Communication and common layer.

They can be found at `hostlib/hostLib/libCommon/infra`. These files are machine generated and hence is not recommended to hand edit them.

3.6.5 Changing logging level

To change the level of logging, the following approaches are valid and based on the need of the problem, they can and should be used.

Full source-code

`hostlib/hostLib/libCommon/infra/nxLog_DefaultConfig.h` can be modified to change logging level. `nxLog_DefaultConfig.h` is self documented.

```
/* See Plug & Trust Middleware Docuemntation --> stack --> Logging
   for more information */

/*
 * - 1 => Enable Debug level logging - for all.
 * - 0 => Disable Debug level logging. This has to be
 *       enabled individually by other logging
 *       header/source files */
#define NX_LOG_ENABLE_DEFAULT_DEBUG 0

/* Same as NX_LOG_ENABLE_DEFAULT_DEBUG but for Info Level */
#define NX_LOG_ENABLE_DEFAULT_INFO 1

/* Same as NX_LOG_ENABLE_DEFAULT_DEBUG but for Warn Level */
#define NX_LOG_ENABLE_DEFAULT_WARN 1

/* Same as NX_LOG_ENABLE_DEFAULT_DEBUG but for Error Level.
 * Ideally, this should always be kept enabled */
#define NX_LOG_ENABLE_DEFAULT_ERROR 1
```

(continues on next page)

(continued from previous page)

```
/* Release - retail build */
#ifdef FLOW_SILENT
#undef NX_LOG_ENABLE_DEFAULT_DEBUG
#undef NX_LOG_ENABLE_DEFAULT_INFO
#undef NX_LOG_ENABLE_DEFAULT_WARN
#undef NX_LOG_ENABLE_DEFAULT_ERROR

#define NX_LOG_ENABLE_DEFAULT_DEBUG 0
#define NX_LOG_ENABLE_DEFAULT_INFO 0
#define NX_LOG_ENABLE_DEFAULT_WARN 0
#define NX_LOG_ENABLE_DEFAULT_ERROR 0
#endif
```

Logging at component level

For example, changing logging level of App, hostlib/hostLib/libCommon/infra/nxLog_App.h can be updated. As shown below, in nxLog_App.h, the values of NX_LOG_ENABLE_APP_INFO, etc. can be updated.

```
/* If source file, or nxLog_Config.h has not set it, set these defines
 *
 * Do not #undef these values, rather set to 0/1. This way we can
 * jump to definition and avoid plain-old-text-search to jump to
 * undef. */

#ifndef NX_LOG_ENABLE_APP_DEBUG
#   define NX_LOG_ENABLE_APP_DEBUG (NX_LOG_ENABLE_DEFAULT_DEBUG)
#endif
#ifndef NX_LOG_ENABLE_APP_INFO
#   define NX_LOG_ENABLE_APP_INFO (NX_LOG_ENABLE_APP_DEBUG + NX_LOG_ENABLE_DEFAULT_
↪INFO)
#endif
#ifndef NX_LOG_ENABLE_APP_WARN
#   define NX_LOG_ENABLE_APP_WARN (NX_LOG_ENABLE_APP_INFO + NX_LOG_ENABLE_DEFAULT_
↪WARN)
#endif
#ifndef NX_LOG_ENABLE_APP_ERROR
#   define NX_LOG_ENABLE_APP_ERROR (NX_LOG_ENABLE_APP_WARN + NX_LOG_ENABLE_DEFAULT_
↪ERROR)
#endif
```

Individual files

Rather than applying logging levels to full stack, if the need is to set the logging level in individual files, the individual source file can set required defined before including the respective log file.

e.g. The below lines will set log level to maximum:

```
#define NX_LOG_ENABLE_APP_DEBUG 1
#include <nxLog_App.h>
```

e.g. The below lines will set log level to just error:

```
#define NX_LOG_ENABLE_APP_DEBUG 0
#define NX_LOG_ENABLE_APP_INFO 0
```

(continues on next page)

(continued from previous page)

```
#define NX_LOG_ENABLE_APP_WARN 0
#define NX_LOG_ENABLE_APP_ERROR 1
#include <nxLog_App.h>
```

nxLog_App.h and _APP_ needs to be replaced with respective names as per the list in *Logging Header Files*.

3.7 Using Platform SCP Keys from File System

Warning: Keeping keys in plain on file system is not secure. This mechanism is just to test quick prototyping / testing.

Note:

- This is valid only for hosts with filesystem access e.g. : Windows/Linux
- *CLI Tool* does not use this mechanism.

Using this mechanism, pre-compiled windows/linux demo examples can pick up platform SCP Keys from file system.

- 1) Create a file as set by EX_SSS_SCP03_FILE_PATH

You can over-ride value for this variable in sss/ex/inc/ex_sss_scp03_keys.h

For Android

```
#define EX_SSS_SCP03_FILE_DIR "/data/vendor/SE05x/"
#define EX_SSS_SCP03_FILE_PATH EX_SSS_SCP03_FILE_DIR "plain_scp.txt"
```

For Linux

```
#define EX_SSS_SCP03_FILE_DIR "/tmp/SE05X/"
#define EX_SSS_SCP03_FILE_PATH EX_SSS_SCP03_FILE_DIR "plain_scp.txt"
```

For Windows

```
#define EX_SSS_SCP03_FILE_DIR "C:\\npx\\SE05X\\"
#define EX_SSS_SCP03_FILE_PATH EX_SSS_SCP03_FILE_DIR "plain_scp.txt"
```

- 2) Let us assume the Platform SCP03 keys provisioned in SE050 are as follows

- ENC is 35C29245895EA34F6136155F8209D6CD
- MAC is AF172D5D54F7C0D5C10A05B9F1207F78
- DEK is A2BC8438BF77015B361A4425F239FA29

The format of a reference file is as below:

```
# This is a comment, empty lines and comment lines allowed.
ENC 35C29245895EA34F6136155F8209D6CD # Trailing comment
MAC AF172D5D54F7C0D5C10A05B9F1207F78 # Optional trailing comment
DEK A2BC8438BF77015B361A4425F239FA29 # Optional trailing comment
```

The Default Platform SCP keys for ease of use configurations are present in <https://www.nxp.com/docs/en/application-note/AN12436.pdf>

3.7.1 How to Run examples with Platform SCP03 keys

Once the `plain_scp.txt` file is filled with the correct SCP keys for the sample, run any example e.g.: *ECC Example*. The example will automatically pick up the keys from the file at this location, if the file exists. If the file does not exist, it uses keys from pre-compiled values in the example.

3.8 Auth Objects

There are 3 types of Auth Objects:

- *Auth Objects : UserID* : Sometimes referred to as Identifier or password
- *Auth Objects : AESKey* : Uses AES Key as authentication
- *Auth Objects : ECCKey* : Uses ECC Key for authentication

These are needed to have an authenticated session to the Applet. Also, in SE05X, access to many objects can be controlled via policies. Many of these policies are related to which Auth Object was used for the session to the SE05X.

3.9 Auth Objects : UserID

As user ID is kind of Symmetric Identifier that is used to authenticate a session.

3.9.1 User ID - Provisioning / Injection

To *provision / inject* the key, the process is like this:

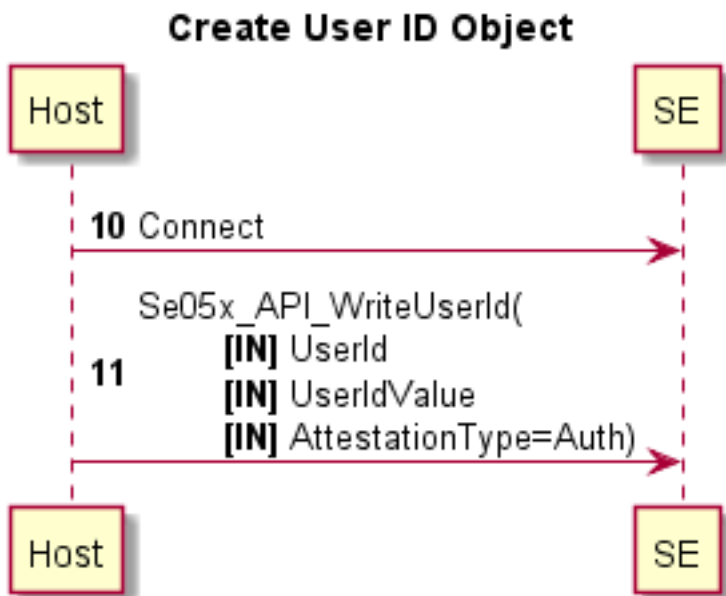


Table 1: Steps to provision

Step	Operation
10	We establish physical connection to SE
11	We create a UserId object, <i>Attestation Type</i> is Auth

3.9.2 User ID - Use for connection / authentication

To *use* the key, the process is like this:

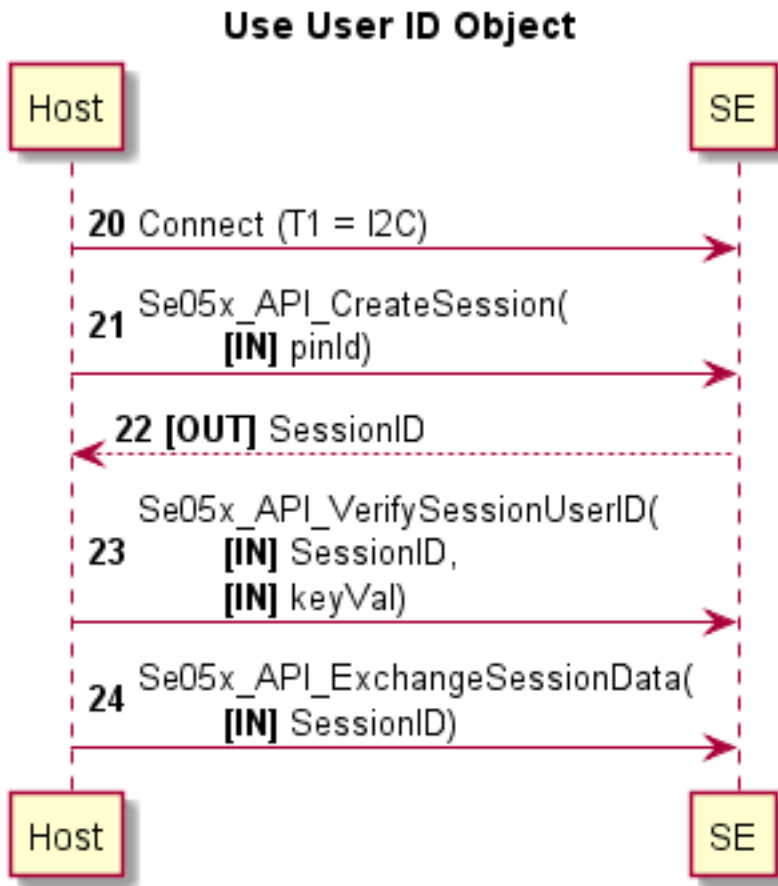


Table 2: Steps

Step	Operation
20	Host establishes physical connection to SE
21	Host calls <code>Se05x_API_CreateSession()</code> and use the 32bit id of UserId that we are going to use.
22	As a part of <code>Se05x_API_CreateSession()</code> API, Applet returns an 8 byte Session ID. We use this in future communication with the SE.
23	Host calls <code>Se05x_API_VerifySessionUserID()</code> . At this point, we pass the Value that we are going to use. (Host must already know the value of the PIN that is used/chosen in step 21.)
24	Finally, Host calls <code>Se05x_API_ExchangeSessionData()</code> API

3.9.3 User ID - Applet Spec Notes

From APDU Spec:

3.2.1.9 UserID

A UserID object is a byte array that holds a value that is linked to a user.

UserID objects can only be created as Authentication object. By default, the maximum number of allowed authentication attempts is set to 255.

Length = 1 up to 16 bytes

3.10 Auth Objects : AESKey

Applet SCP03 is applet level secure channel protocol, which involved single symmetric key as static ENC, MAC, DEC key. First applet scp session is created using auth object provisioned in the SE, and then it follows the standard SCP03 protocol — Initialized Update and External Authenticate.

3.11 Auth Objects : ECKey

ECKey is secure channel protocol tailored for secured authentication and communication between a Host and a connected SE.

Please contact NXP CAS/FAE for the specification of ECKey.

The Secure Channel Protocol consists of two logical phases:

- 1) Authentication phase
- 2) Secure messaging phase

3.11.1 ECKey - Keys Used

The table below gives an overview of the required keys and their presence at SE and Host as required for the ECKey setup, authentication phase

Table 3: Auth Keys

key	SE	Host	Purpose
SK.SE.ECKA	Yes		Static SE key pair for Key Agreement Private key
PK.SE.ECKA	Yes	Yes	Static SE key pair for Key Agreement Public key
SK.Host.ECDSA		Yes	Host signing key pair Private key
PK.Host.ECDSA	Yes		Host signing key pair Private key
eSK.Host.ECKA		Yes	Ephemeral private key of the Host used for key agreement
ePK.Host.ECKA		Yes	Ephemeral public key of the Host used for key agreement

3.11.2 ECKey - Use for connection / authentication

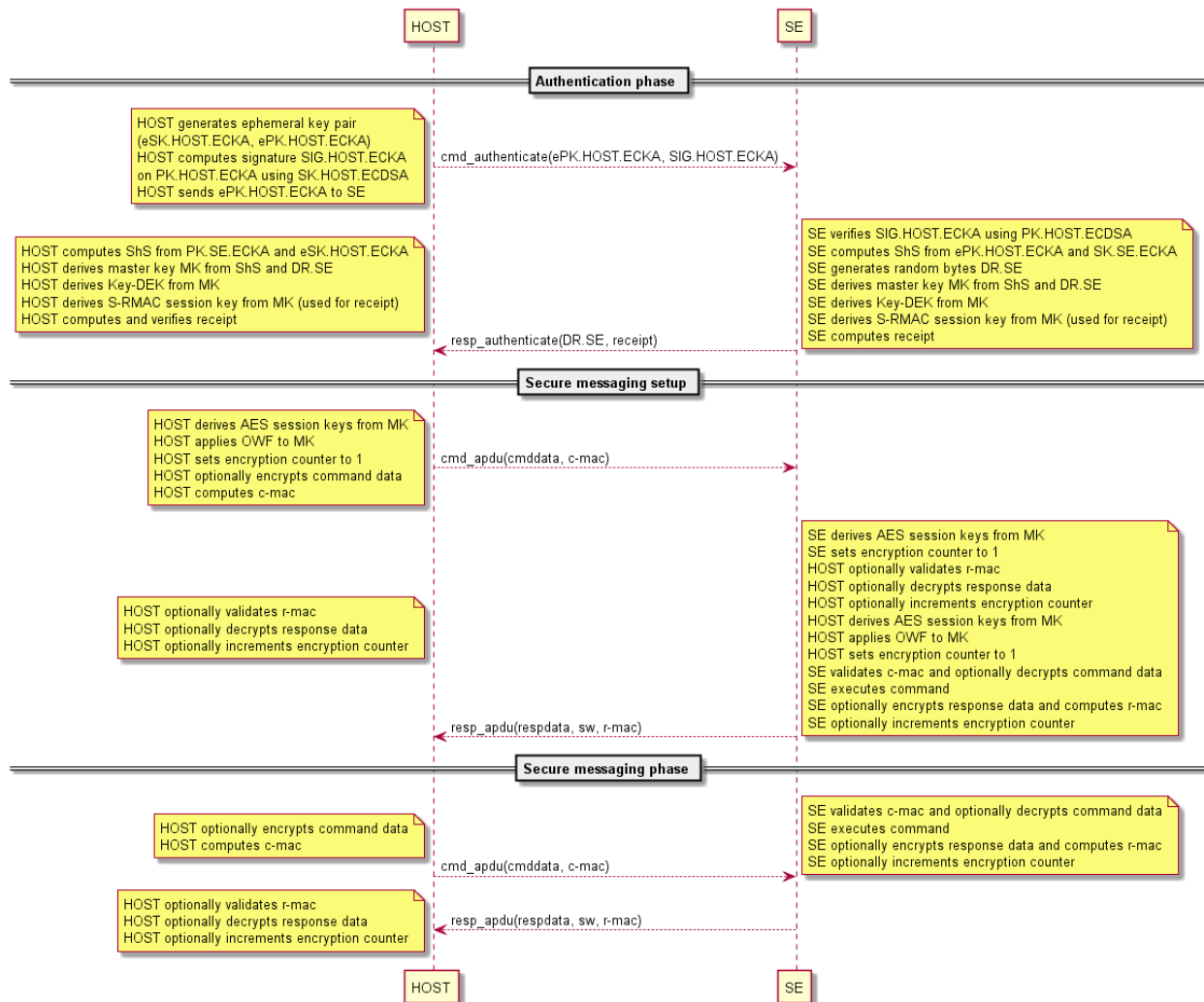
Authentication Phase:

- 1) In the Secure Channel authentication phase, a Host-generated ephemeral key pair, the static SE key pair and SE-generated random data are used to compute a shared master secret
- 2) The Host generates an ephemeral ECC key pair and exchanges the public key component ePK.Host.ECKA with the SE. The ePK.Host.ECKA is signed to prove its authenticity.
- 3) Both the Host and the SE compute the shared secret ShS from (eSK.Host.ECKA , PK.SE.ECKA) and (SK.SE.ECKA, ePK.Host.ECKA) respectively.
- 4) The SE generates random bytes DR.SE and exchanges this with the Host.
- 5) Both the Host and the SE compute the shared master secret MK from the shared secret ShS and random bytes DR.SE.
- 6) **Optionally:**
 - A) Both Host and SE compute the Key-DEK.
 - B) Both Host and SE compute the S-RMAC session key (used for the receipt).
 - C) Both Host and SE compute the receipt.
 - D) The Host verifies the receipt.

Secure Messaging phase:

- 1) In the Secure Channel secure messaging phase, first a setup is performed, where the shared master secret is used to compute the AES session keys (S-ENC, S-MAC and S-RMAC) and initialize the encryption counter
- 2) Both the Host and the SE compute the AES session keys from MK.
- 3) Both the Host and the SE apply an OWF to MK. Note that this is performed after the validation of the command's C-MAC (to exclude DOS attacks on the SE).
- 4) Both the Host and the SE initialize the encryption counter to 1 for the first command/response with C-DECRYPTION or R-ENCRYPTION.
- 5) After the secure messaging setup has been performed, the AES session keys are employed to realize SCP03 secure messaging between Host and SE
- 6) Command APDUs are MACed.
- 7) Response APDUs are optionally MACed.
- 8) Command and response APDUs are optionally encrypted. For each command/response with C-DECRYPTION or R-ENCRYPTION, the encryption counter is incremented.

The phases inclusive the optional Key-DEK and receipt are shown in the figure below



3.12 Key Id Range and Purpose

Purpose		Range
Customer keys	Start	0x00000001
	End	0x7BFFFFFF
AKM Dynamic	Start	0x7C000000
	End	0x7CFFFFFF
Middleware Use Cases and Demos	Start	0x7D000000
	End	0x7DFFFFFF
Applet Reserved	Start	0x7FFF0000
	End	0x7FFFFFFF
Managed by EdgeLock 2GO for different cloud onboarding	Start	0x80000000
	End	0xEEFFFFFF
Middleware testing	Start	0xEF000000
	End	0xEFFFFFFF
Required for EdgeLock 2GO	Start	0xF0000000
	End	0xFFFFFFFF

3.13 Trust provisioned KeyIDs

Trust Provisioned object type	KeyID
ECC-256	
Device Key	0xF0000100
Device Certificate	0xF0000101
Gateway Key	0xF0000102
Gateway Certificate	0xF0000103
RSA-2K	
Device Key	0xF0000110
Device Certificate	0xF0000111
Gateway Key	0xF0000112
Gateway Certificate	0xF0000113
RSA-4K	
Device Key	0xF0000120
Device Certificate	0xF0000121
Gateway Key	0xF0000122
Gateway Certificate	0xF0000123

BUILDING / COMPILING

The Plug & Trust Middleware uses *CMake* for compiling / building.

4.1 Windows Build

4.1.1 Prerequisite

- Visual studio installed
- Python 3 32 bit installed

4.1.2 Create Build files

- Use `<SE05X_root_folder>/simw-top/scripts/create_cmake_projects.py` to generate the build files, Run as

```
@REM Setup environment variables and PATH
call env_setup.bat

@REM Use CMake to generate .sln files
python create_cmake_projects.py
```

- Build files are generated at `<SE05X_root_folder>/simw-top_build/`
- Use the visual studio solution at `<SE05X_root_folder>/simw-top_build/se_x86/PlugAndTrustMW.sln` to build sample examples and demo examples

4.1.3 SSS Examples

- Sample examples can be found at `<SE05X_root_folder>/simw-top_build/se_x86/bin`
- Run the example as

```
<example_name>.exe <COM_PORT>
```

Refer *Demo and Examples* for details on these examples and for running cloud/tls demo applications

4.2 Import MCUXPresso projects from SDK

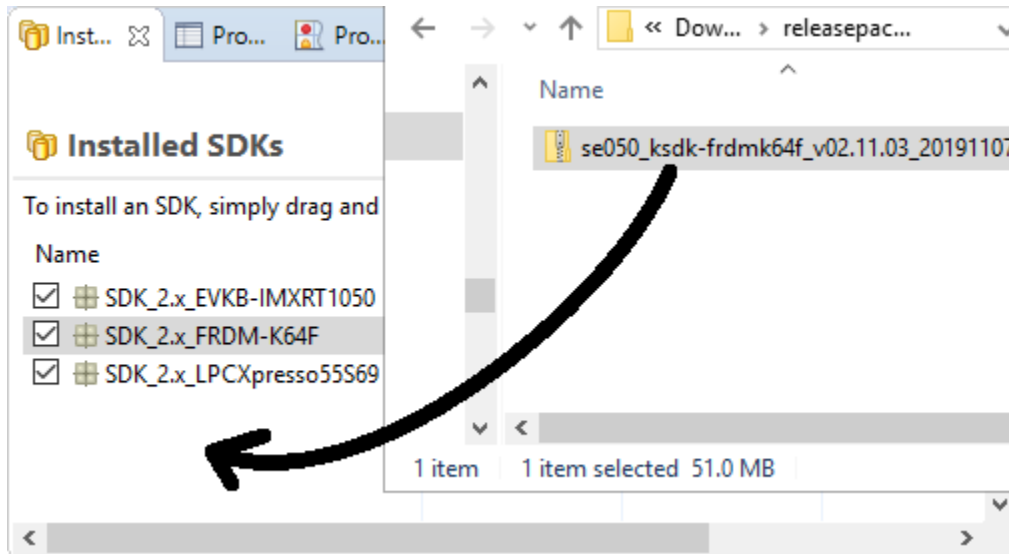
With Plug & Trust middleware, MCUXPresso SDKs are supplied for supported boards. Projects can be imported/executed from these SDKs.

4.2.1 Prerequisite

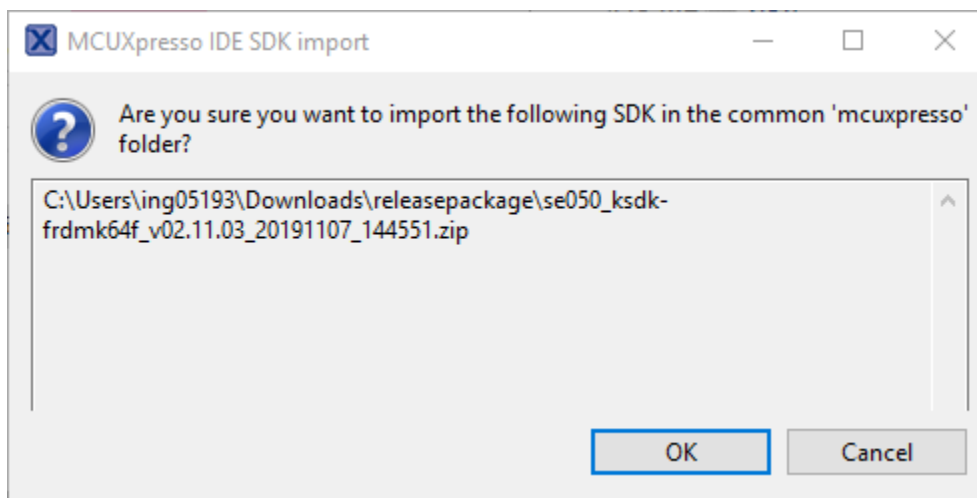
- MCUXpresso installed. Refer to *Setting up MCUXPresso IDE* for details on installation
- Plug & Trust Middleware SE050 MCUXPresso SDK Package is downloaded and available

4.2.2 Importing an example

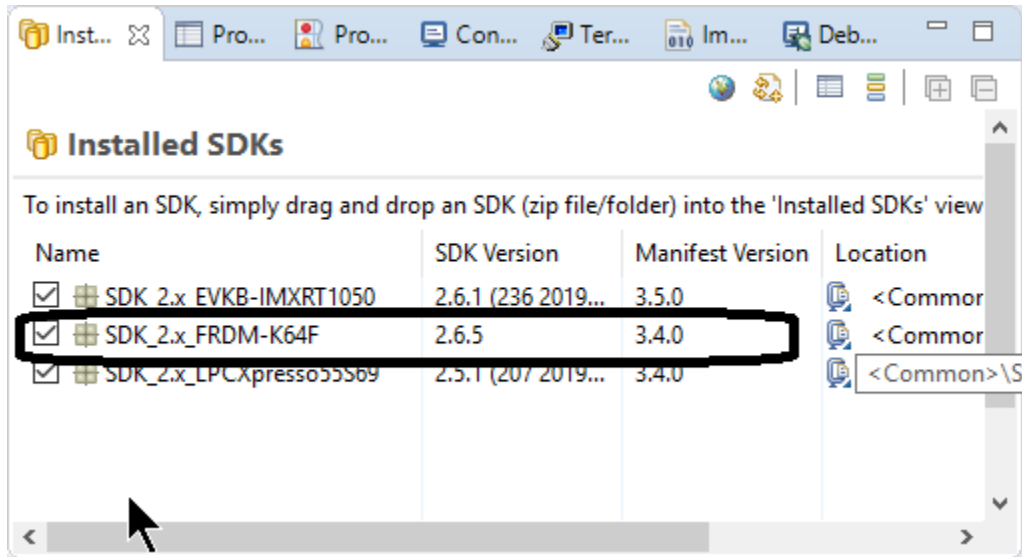
- Drag and drop SDK to “Install SDKs” windows of MCUXPresso. Even if you have older SDK, you can drag and drop. MCUXPresso will take the latest SDK based on version number.



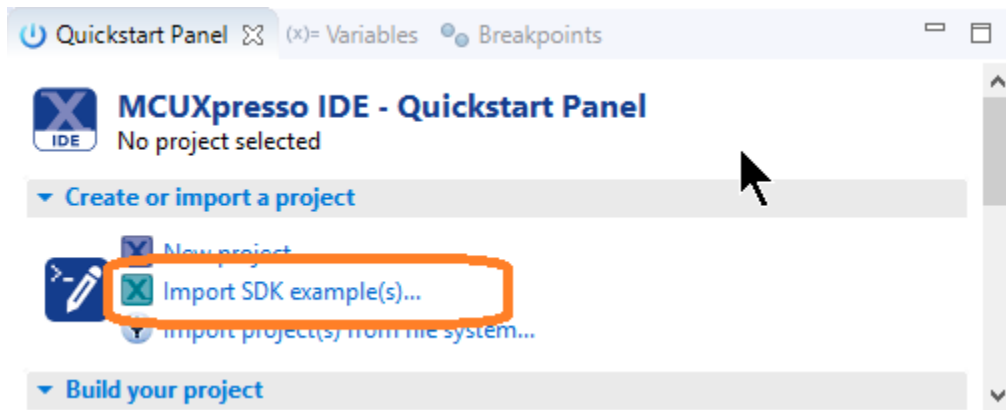
- Accept the request to import the SDK.



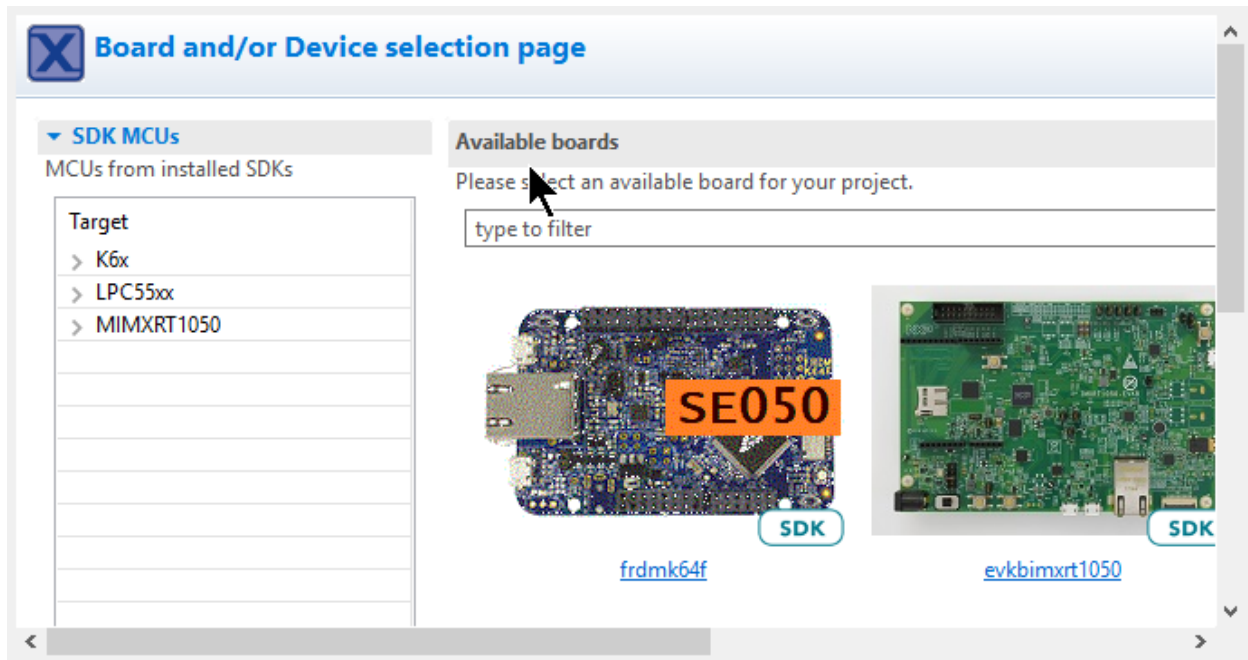
- You should see a new / updated entry. In this case version 2.6.5. Based on the release, this version may change. If you have Newer version of SDK, and you would like to use this version, you must delete the old SDK.




- From the “Quickstart Panel” -> “Import SDK Example(s)”



- You must see K64F with SE050 marked. For other SDKs, same convention follows. If you do not see SE050 there, it means you have newer version of the SDK and you must delete them.



- When you import and click next, you will see support examples for that board.
- Select the example you would like to run/test. In this case, we go for “minimal”


Import projects

Project name prefix:
Project name suffix:

☒ Use default location

Location:

Project Type
☒ C Project
☐ C++ Project
☐ C Static Library
☐ C++ Static Library

Project Options
☐ SDK Debug Console
☐ Semihost
☒ UART
☐ Example default
☒ Copy sources
☒ Import other files

Examples

Name	Description	Version
<input type="checkbox"/> se_SE05x_cloud_aws	This demo demonstrates connection to AWS IoT Console using pre-provisione...	
<input type="checkbox"/> se_SE05x_cloud_azure	This demo demonstrates connection to Azure IoT Hub using pre-provisioned d...	
<input type="checkbox"/> se_SE05x_cloud_gcp	This demo demonstrates connection to Google Cloud Platform using pre-prov...	
<input type="checkbox"/> se_SE05x_cloud_ibm	This demo demonstrates connection to IBM Watson IoT platform using pre-pr...	
<input type="checkbox"/> se_SE05x_ex_ecc	This example does a elliptic curve cryptography signing and verify operation.	
<input type="checkbox"/> se_SE05x_ex_hkdf	This example does a HMAC Key derivation operation based on the info and salt.	
<input type="checkbox"/> se_SE05x_ex_md	This example does a Message Digest hashing operation.	
<input type="checkbox"/> se_SE05x_ex_rsa	This example does a RSA signing and verify operation.	
<input type="checkbox"/> se_SE05x_ex_symmetric	This example does a symmetric cryptography AES encryption and decryption o...	
<input type="checkbox"/> se_hostlib_mainA71CH	The mainA71CH demo application demonstrates the usage of Secure Module f...	
<input type="checkbox"/> se_hostlib_se05x_ex_i2cMaster	This example reads Accelerometer data via the I2C master interface.	
<input type="checkbox"/> se_hostlib_se05x_get_info	This project can be used to get SE05X platform information.	
<input checked="" type="checkbox"/> se_hostlib_se05x_minimal	This is a bare minimum example for se050. This gets the amount of free memo...	
<input type="checkbox"/> se_hostlib_vcomA71CH	The vcomA71CH demo application allows the board to be used as a bridge bet...	
<input type="checkbox"/> se_hostlib_vcomSE050	The vcomSE050 demo application allows the board to be used as a bridge bet...	

- You can choose different settings if you like to as supported by MCUXPresso

Advanced Settings

C/C++ Library Settings

Set library type (and hosting variant) NewlibNano (nohost)

☐ Redlib: Use floating point version of printf
☐ Redlib: Use character rather than string based printf
☒ Redirect SDK "PRINTF" to C library "printf"
☒ Include semihost HardFault handler

☐ NewlibNano: Use floating point version of printf
☐ NewlibNano: Use floating point version of scanf
☐ Redirect printf/scanf to ITM
☒ Redirect printf/scanf to UART

Hardware settings

Set Floating Point type FPv4-SP (Hard ABI)

MCU C Compiler

Language standard GNU C99 (-std=gnu99)

MCU Linker

☐ Link application to RAM

Memory Configuration

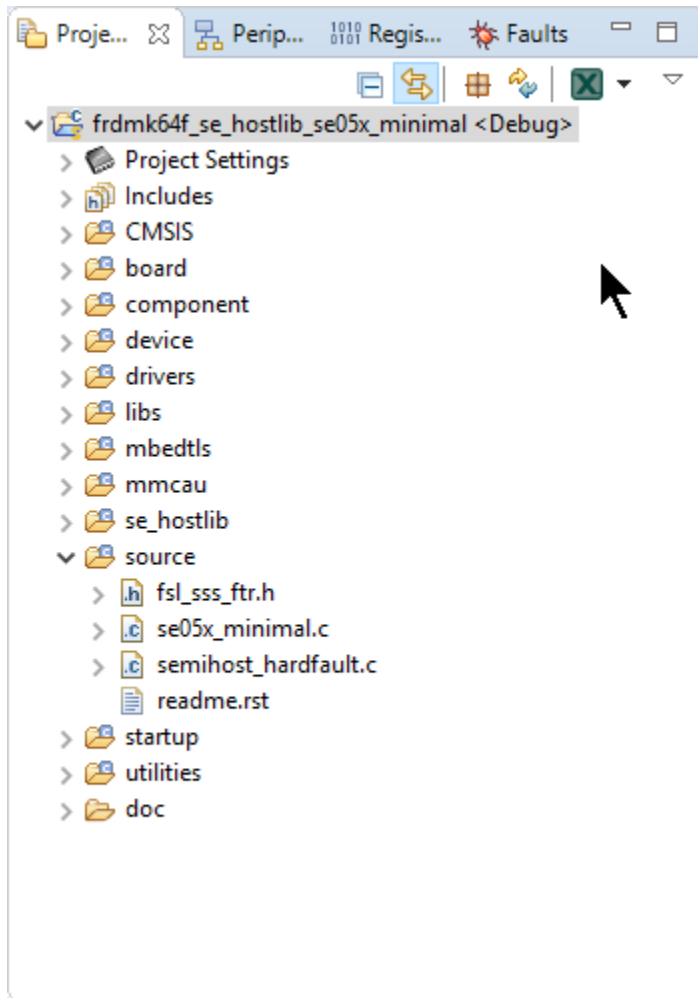
Memory details

Default LinkServer Flash Driver Browse...

Type	Name	Alias	Location	Size	Driver
Flash	PROGRAM_FLASH	Flash	0x0	0x100000	FTFE_4K.cfx
RAM	SRAM_UPPER	RAM	0x20000000	0x30000	
RAM	SRAM_LOWER	RAM2	0x1fff0000	0x10000	
RAM	FLEX_RAM	RAM3	0x14000000	0x1000	

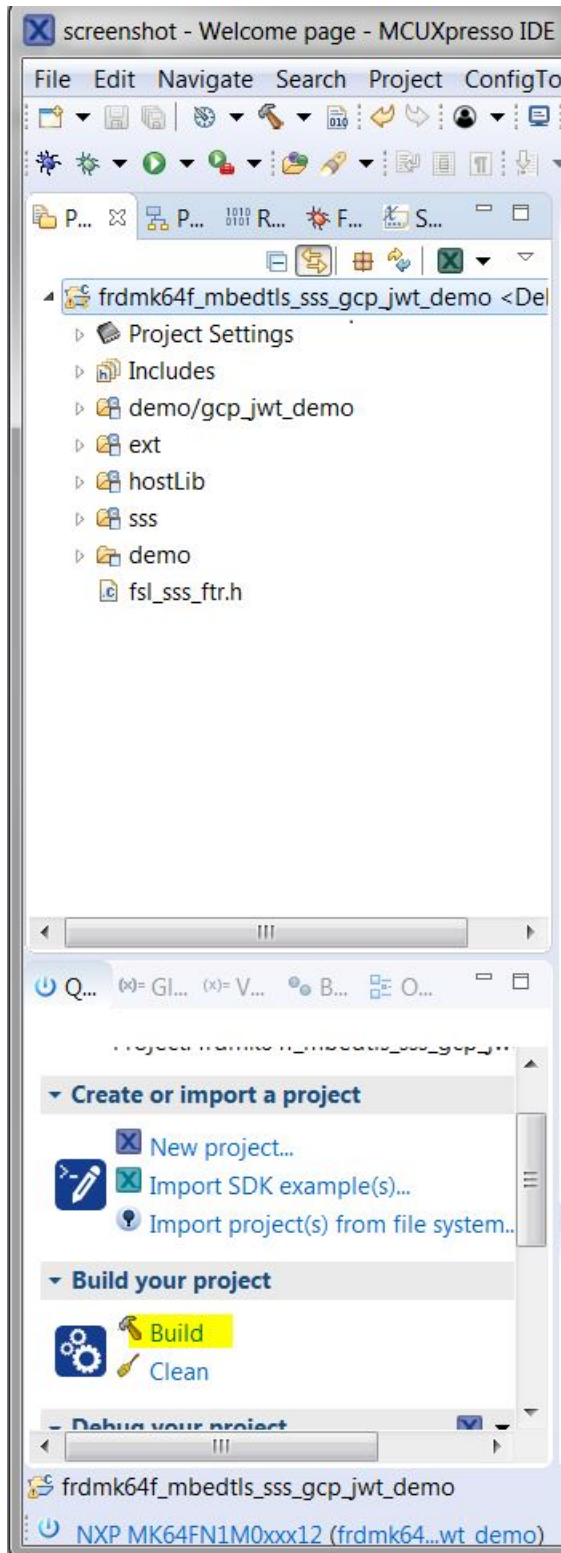
Add Flash Add RAM Split Join Delete Import... Merge... Export... Generate...

- Now the examples are ready to be tested.

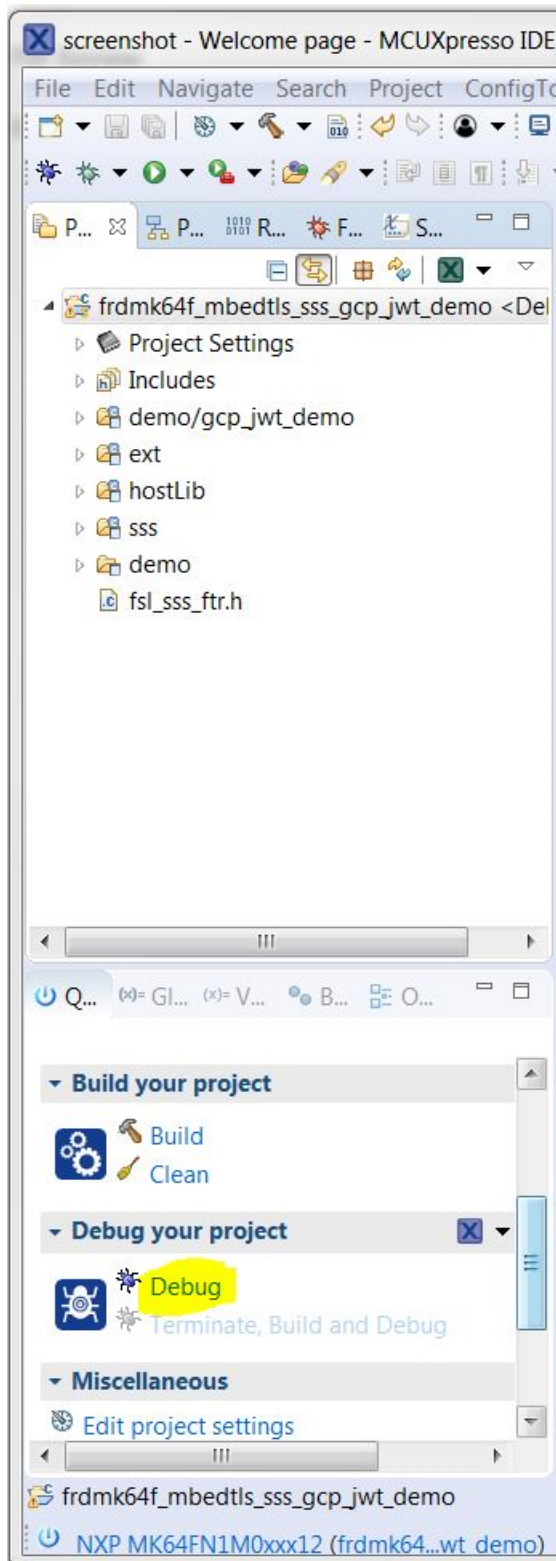


4.2.3 Running / Debugging example

- Click on Build to compile and generate the executables

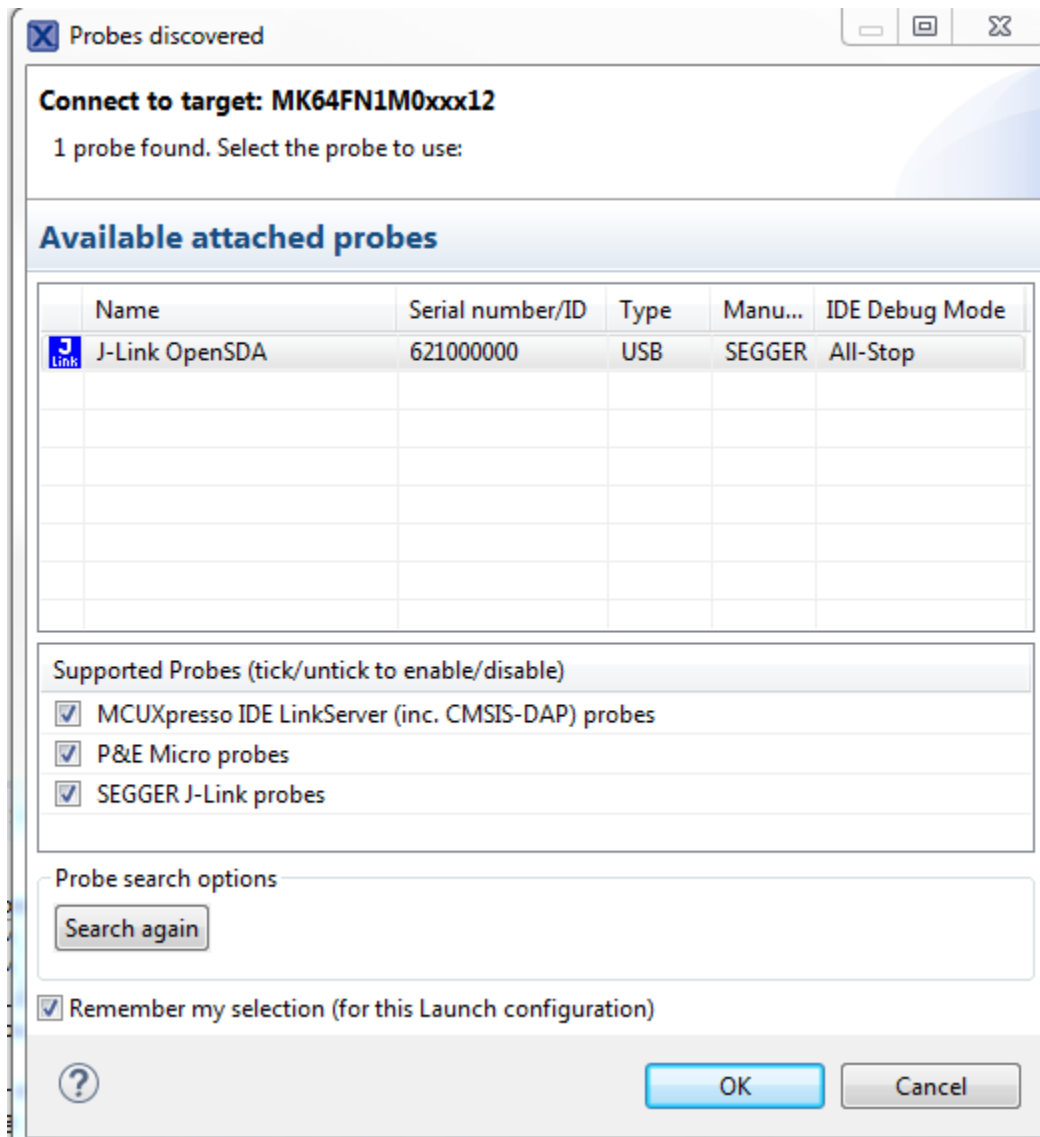


- Click on debug to program the binary into the device

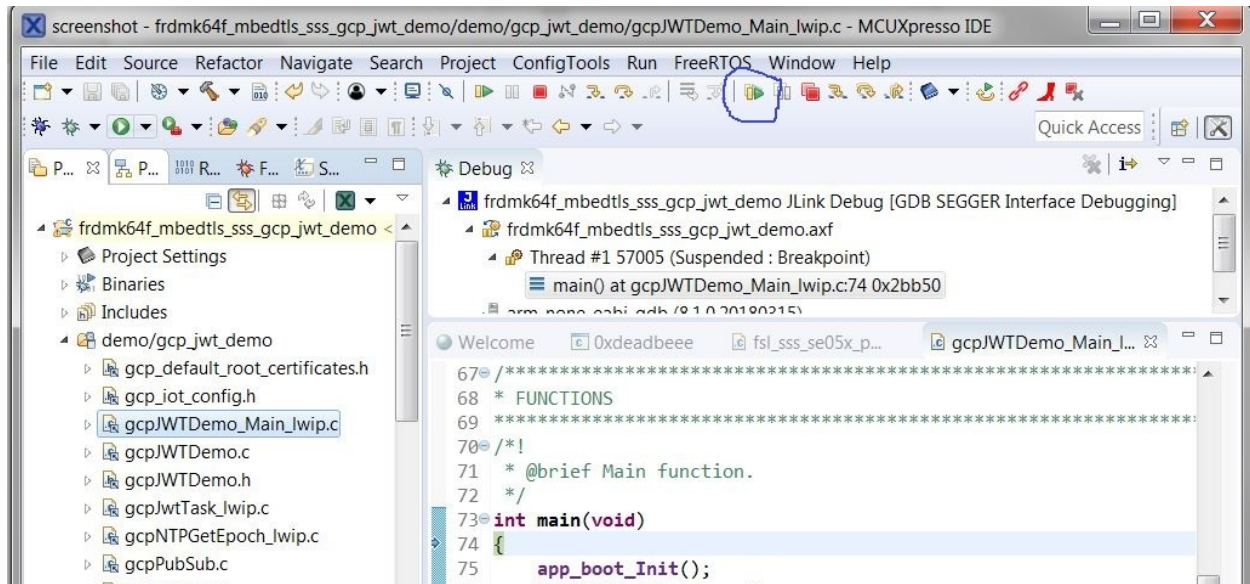


- Select J-Link OpenSDA and click on OK. For more details on OpenSDA, refer to

<https://www.nxp.com/support/developer-resources/run-time-software/kinetis-developer-resources/ides-for-kinetis-mcus/opensda-serial-and-debug-adaptor:OPENSDA>



- Click on the Resume button to start program execution



4.2.4 Logging on console

For UART, a serial terminal application (for e.g. Tera Term) on PC for VCOM serial device needs to be configured as follows

- 115200 baud rate
- 8 data bits
- No parity
- One stop bit
- No flow control

Once the program execution begins, logs are printed on the terminal (e.g. Tera Term) indicating the status of execution and test results. The ATR and the various module versions are printed followed by a SELECT-DONE after which the respective test logs are printed.

4.3 Freedom K64F Build (CMake - Advanced)

For the advanced users and users who are aware of CMake / want to use true potential of CMake, the plug and trust MW supports build system where developers can run exactly the same example on PC/Windows/Linux and embedded targets.

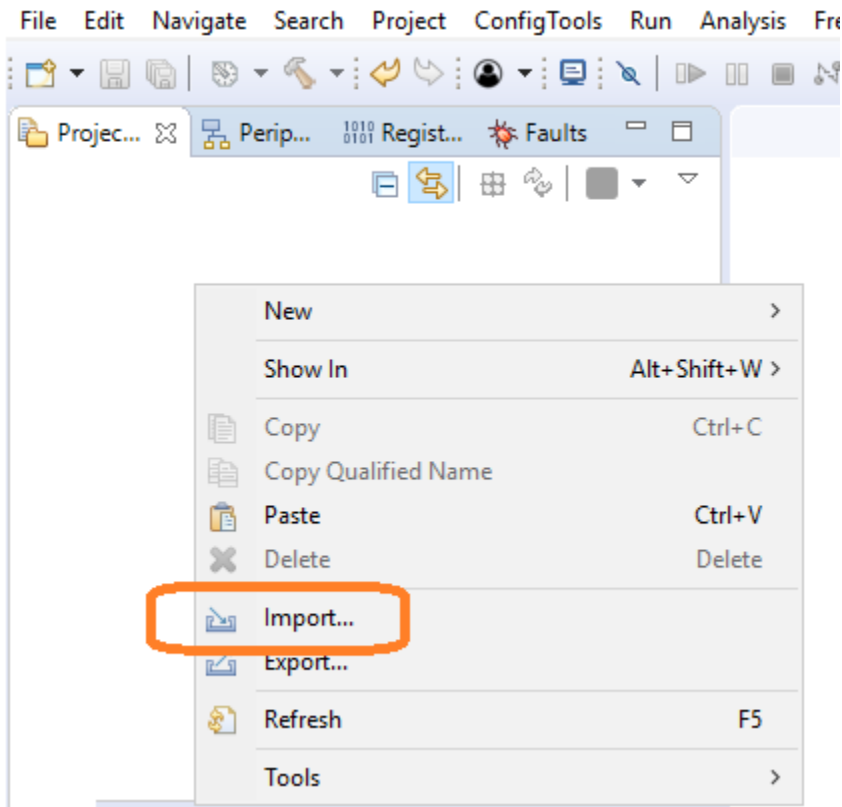
e.g. Using this, developer can develop/extend example on a Windows Machine with visual studio which supports multiple break points, watch points, etc. and then recompile same example for K64F and test it there.

4.3.1 Prerequisite

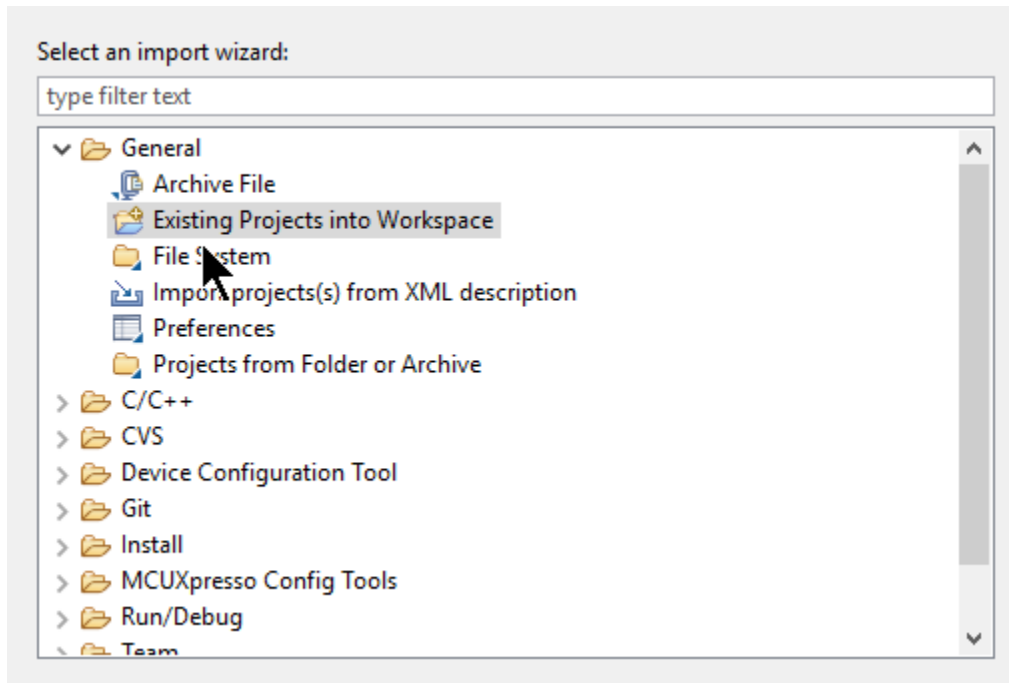
- MCUXpresso installed. Refer to *Setting up MCUXpresso IDE* for details on installation
- CMake is installed
- Python is installed and projects are created using `scripts/create_cmake_projects.py`

4.3.2 Importing the Project

- Import a project using the option provided by MCUXpresso. Do not use the “Quickstart Panel” to import project.

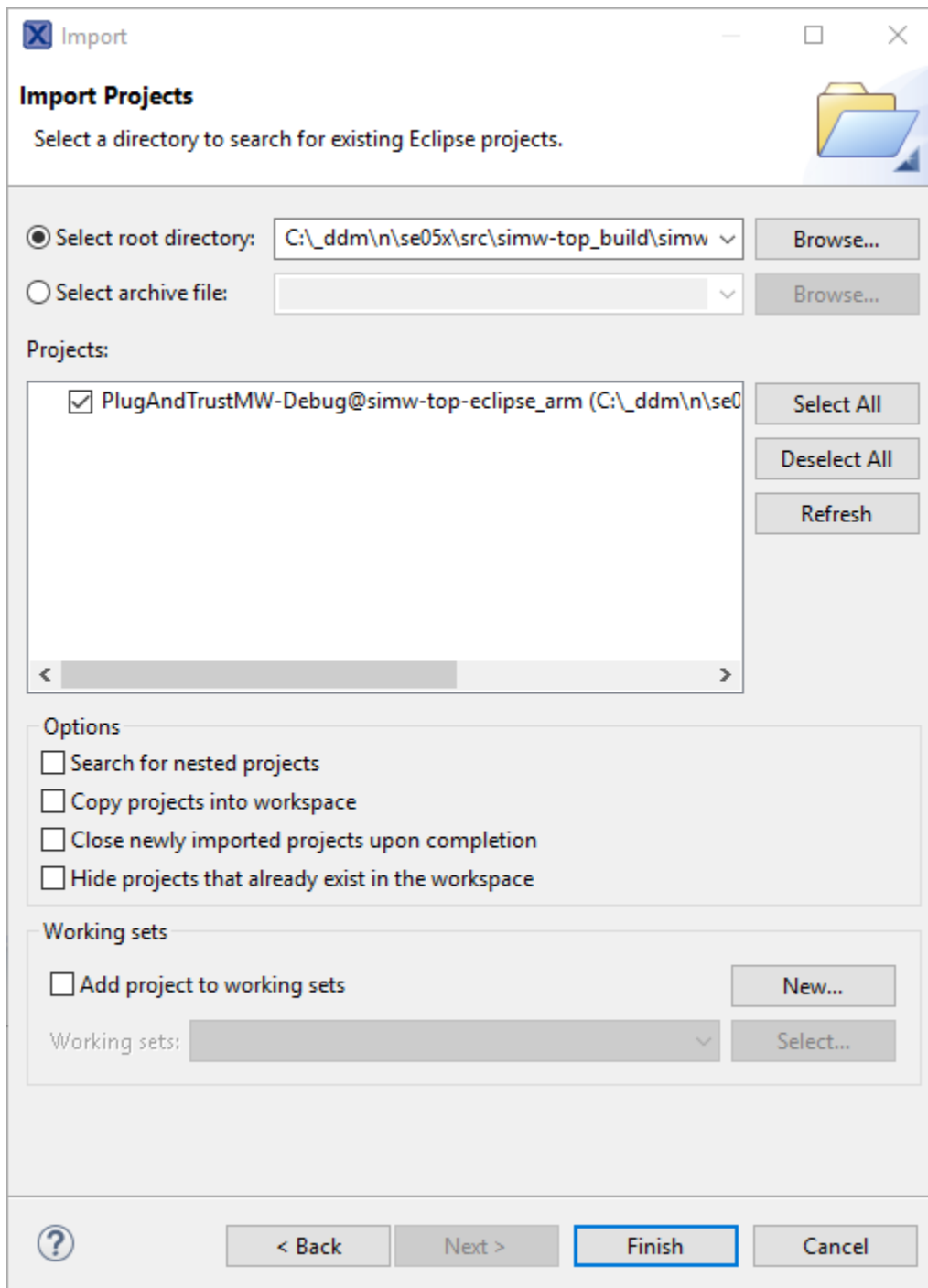


- Import existing projects into Workspace



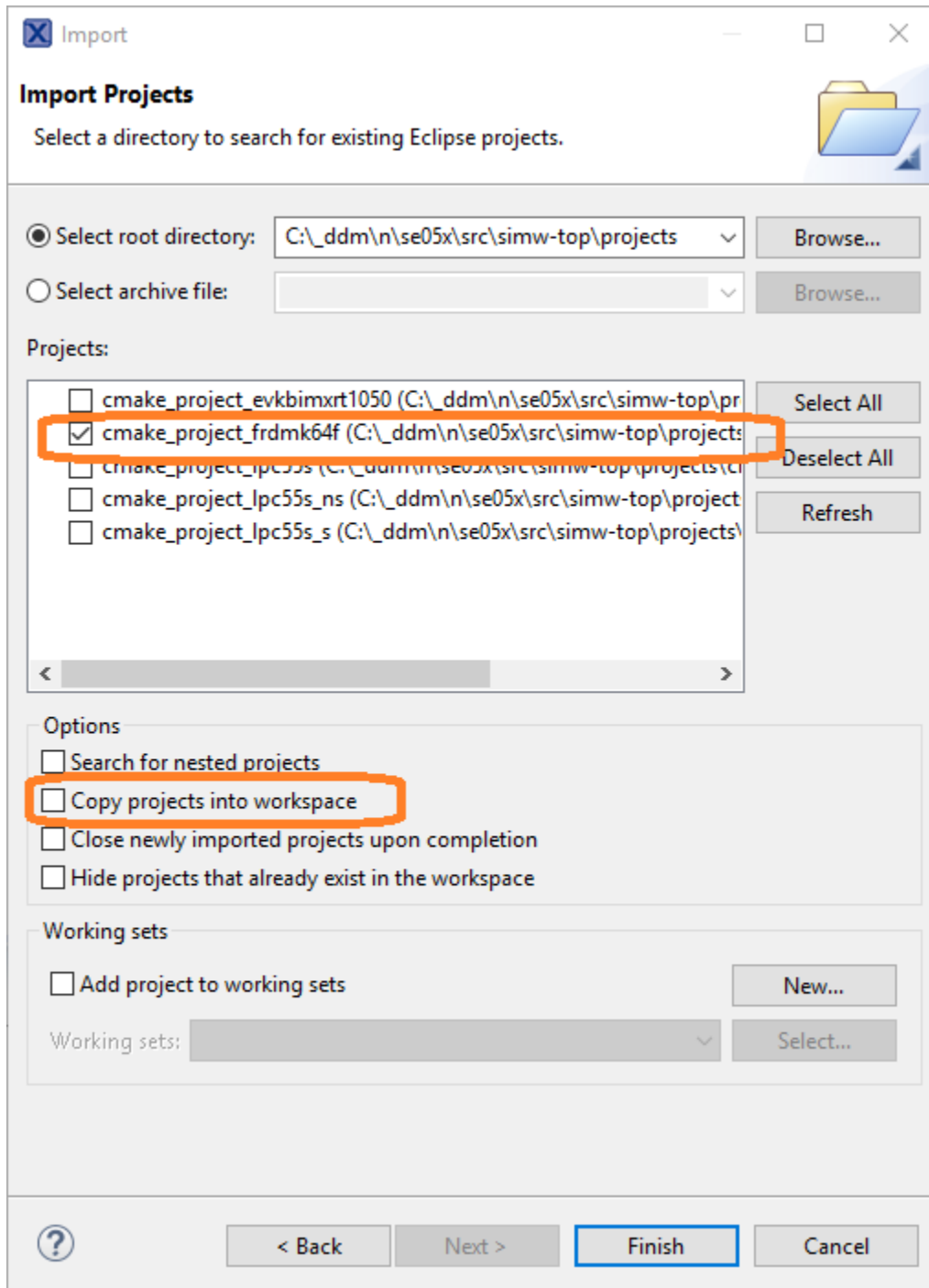
- First import the project generated by `scripts/create_cmake_projects.py`. We are going to use that workarea to build the examples.

Never select “Copy projects into workspace” for this CMake configuration.

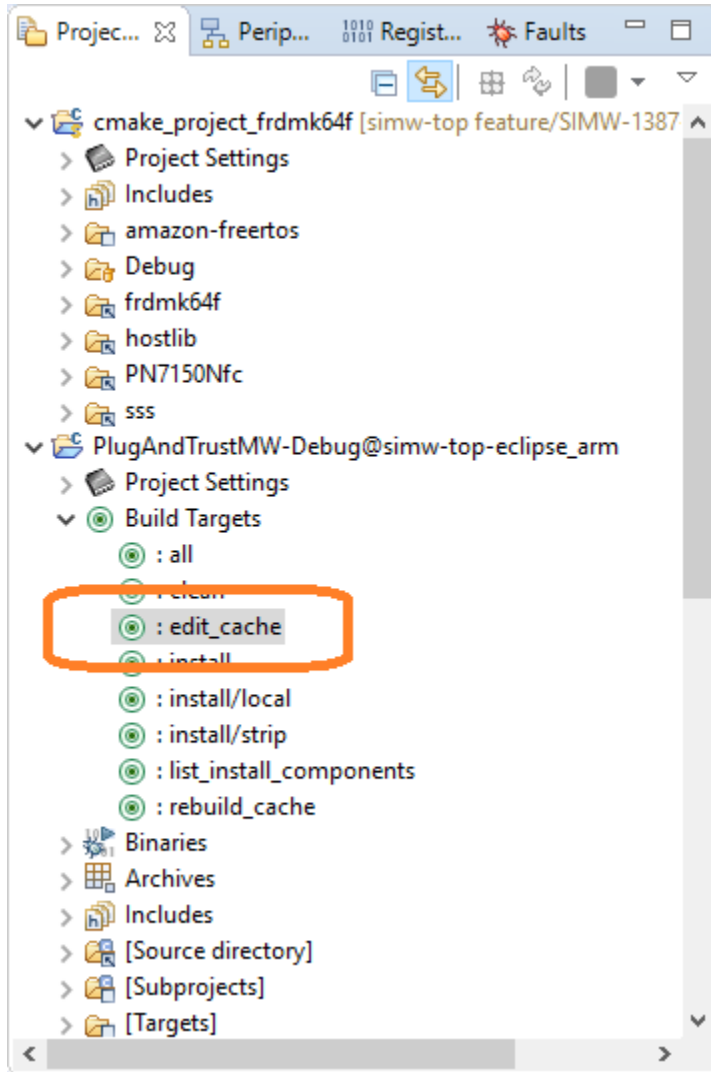


- Then import the project from “projects” directory. Import only the project for your board. In this case we import `cmake_project_frdmk64f`. This project is used to *debug* and *download* to the target. It’s a manually maintained project.

Never select “Copy projects into workspace” for this CMake configuration.



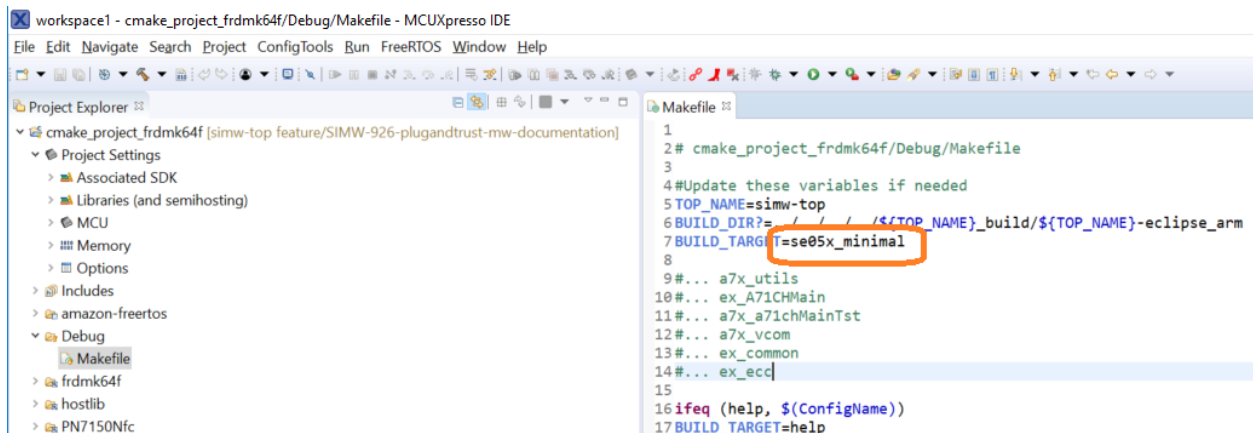
- You may have to run cmake's `edit_cache` to change the configuration of your example. e.g. choice of RTOS, Applet, Board, optimization level, etc.



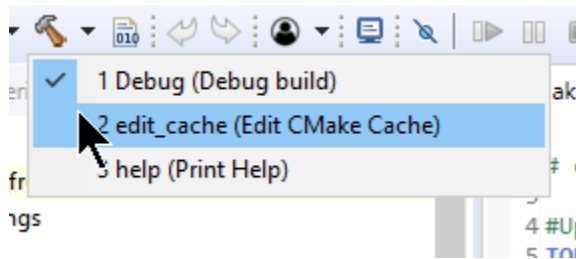
- You would mostly have to change the target that you want to build and download. Modify `cmake_project_frdmk64f/Debug/Makefile`.

As you can see, the default selection is `se05x_minimal`.

You can run the help target of this project as shown in next image to see which projects are supported.

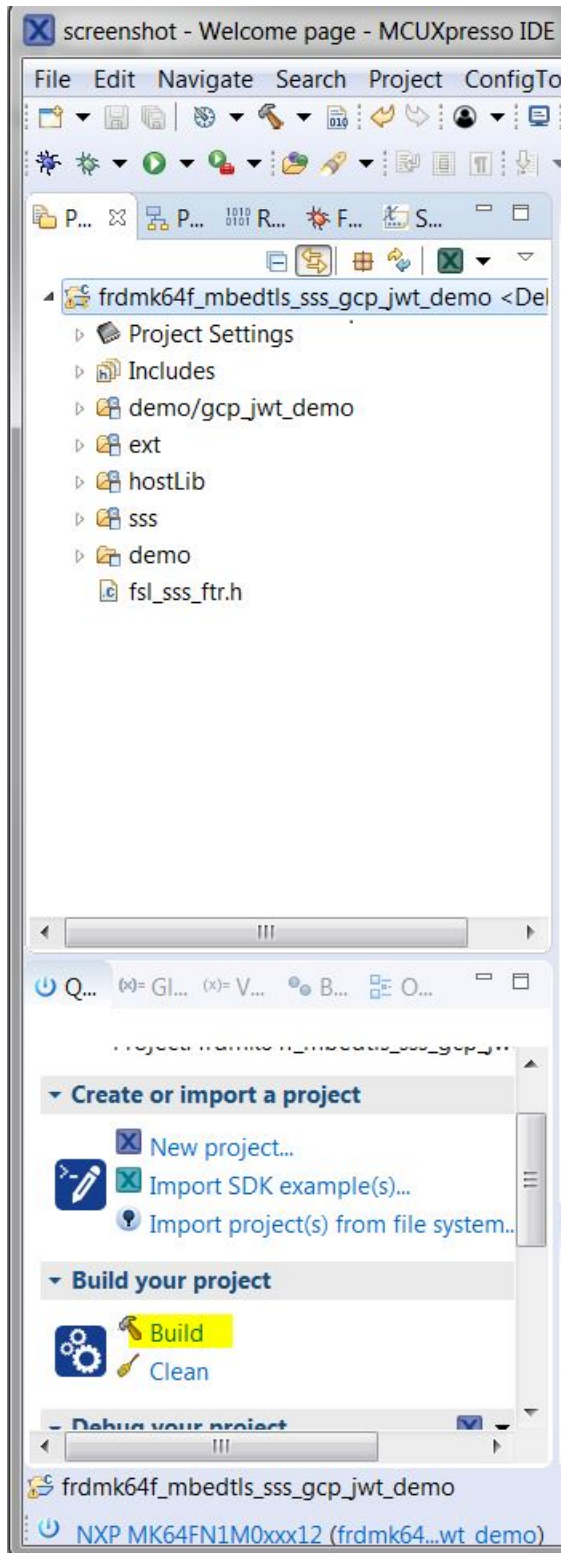


- `cmake_project_frdmk64f` project's settings also supports helpful target to `edit_cache` or list supported projects by that configuration.

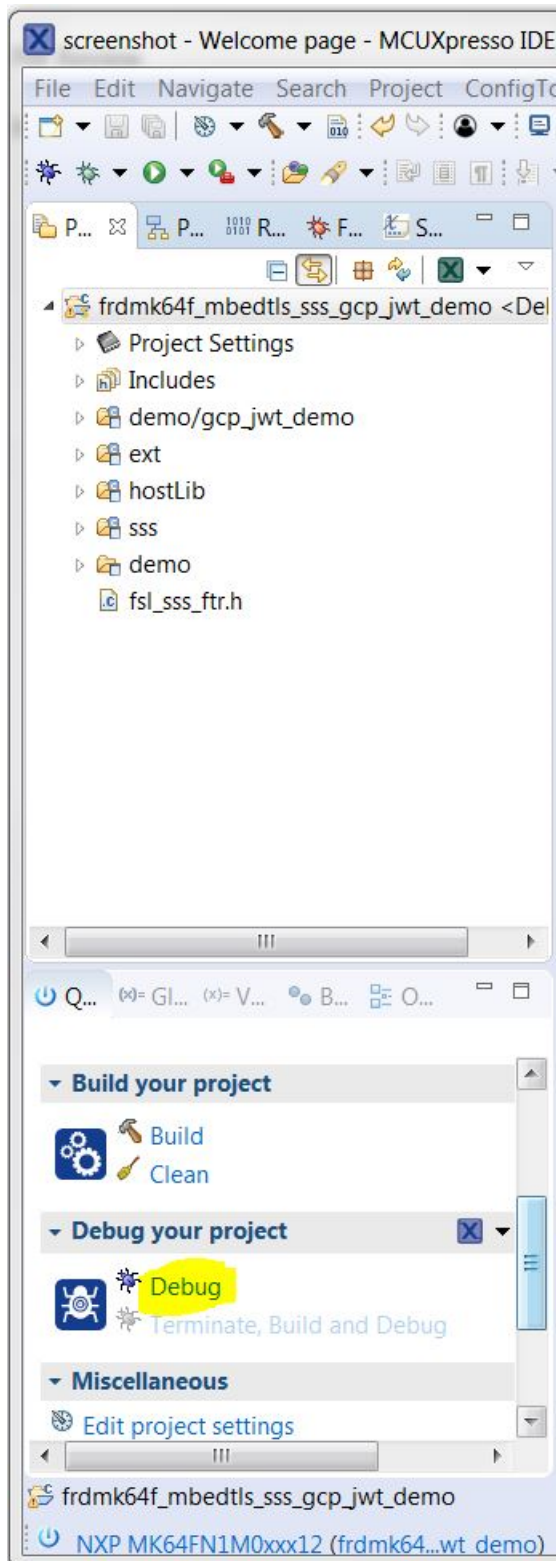


4.3.3 Running / Debugging example

- Click on Build to compile and generate the executables

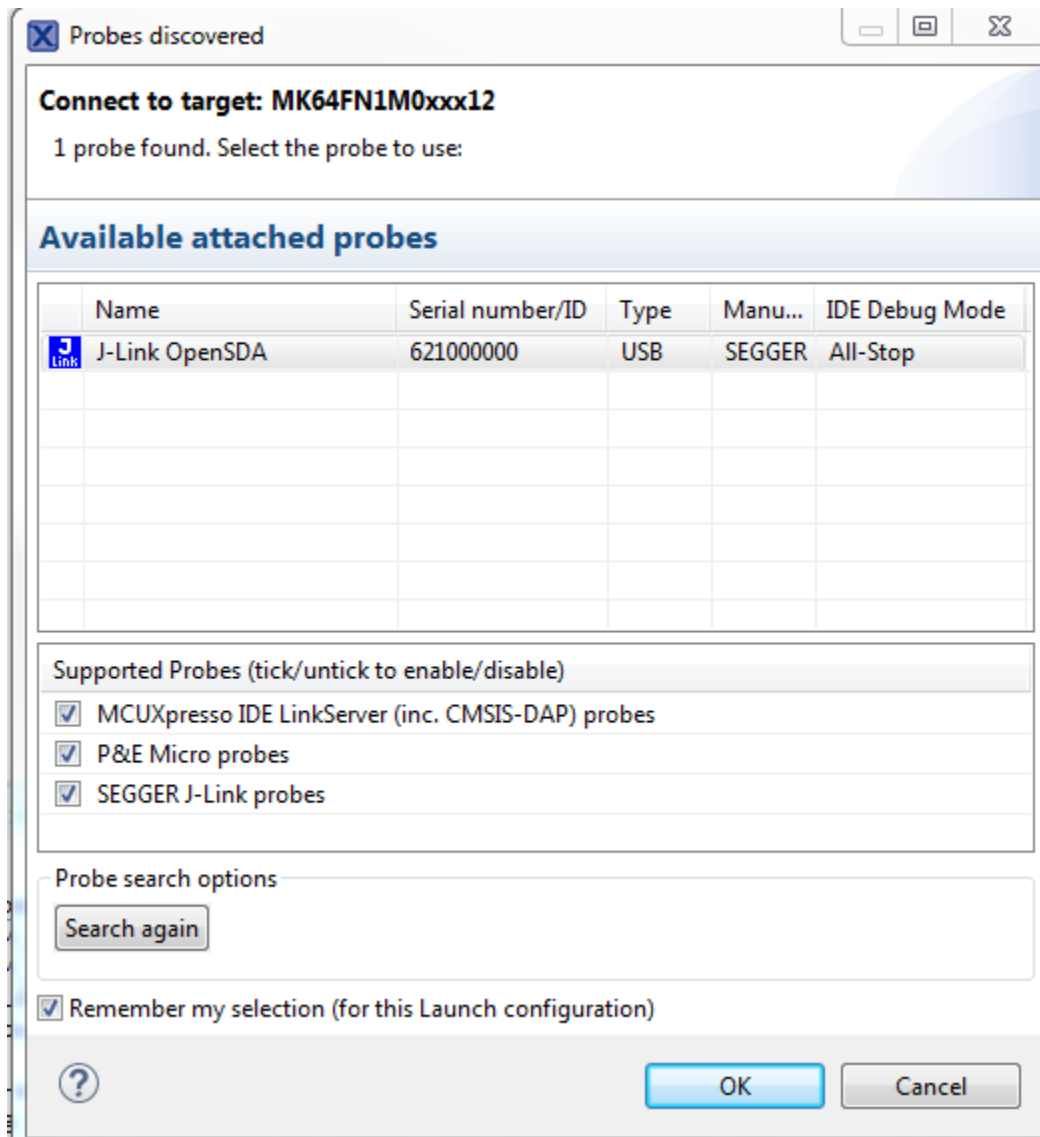


- Click on debug to program the binary into the device

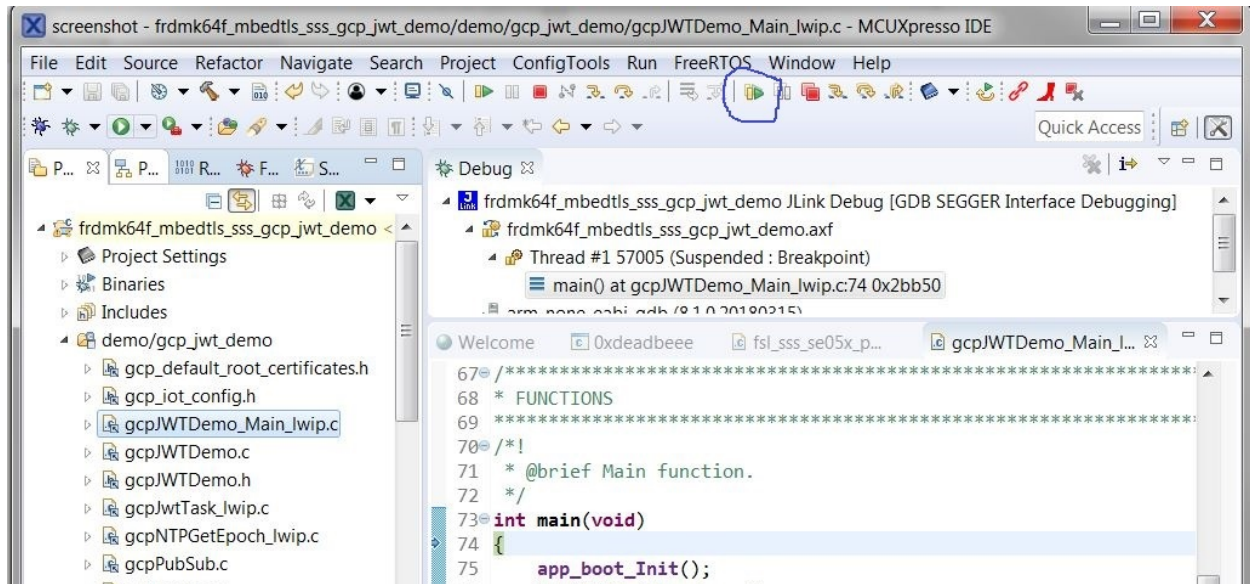


- Select J-Link OpenSDA and click on OK. For more details on OpenSDA, refer to

<https://www.nxp.com/support/developer-resources/run-time-software/kinetis-developer-resources/ides-for-kinetis-mcus/opensda-serial-and-debug-adaptor:OPENSDA>



- Click on the Resume button to start program execution



4.3.4 Logging on console

For UART, a serial terminal application (for e.g. Tera Term) on PC for VCOM serial device needs to be configured as follows

- 115200 baud rate
- 8 data bits
- No parity
- One stop bit
- No flow control

Once the program execution begins, logs are printed on the terminal (e.g. Tera Term) indicating the status of execution and test results. The ATR and the various module versions are printed followed by a SELECT-DONE after which the respective test logs are printed.

4.4 i.MX Linux Build

4.4.1 Prerequisite

Linux should be running on e.g. the MCIMX8M-EVK or MCIMX6UL-EVK development board. Refer to [Setup i.MX 8MQuad - MCIMX8M-EVK](#) for details on preparing the platform.

4.4.2 Build Instructions

- 1) Copy the plug and trust middleware package to the i.MX file system
- 2) Execute the below commands to build and install the se05x libraries on the system:

```
cd simw-top
python scripts/create_cmake_projects.py
cd ../simw-top_build/imx_native_se050_tloi2c
cmake --build .
make install
ldconfig /usr/local/lib
```

Default cmake options are shown below:

Applet	SE050_C
Host	iMXLinux
HostCrypto	OPENSSL
IOT	GCP
OpenSSL	1_1_1
RTOS	Default
SCP	None
SE05X_Auth	None
SMCOM	Tloi2C
WithCodeCoverage	OFF
WithNXPNFCRdLib	OFF
WithSSS_TestCounterPart	ON
WithSharedLIB	OFF
mbedtls_ALT	None

To get the list of enabled cmake options, execute the following command from the build area:

```
cmake -L .
```

- 3) If required cmake options can be changed and libraries can be rebuilt and installed. Refer to [CMake](#)

```
cd simw-top_build/imx_native_se050_tloi2c
ccmake .
```

ccmake (i.e. a text-ui to cmake) is not available by default on the Yocto distribution for i.MX.

An alternative approach to change cmake options is to set them on the command line.

Explicitly overruling cmake options (in this case changing SE05X_Auth to UserID):

```
cd simw-top_build/imx_native_se050_tloi2c
cmake -DSE05X_Auth=UserID .
cmake --build .
```

An easier approach is to use the script `simw-top/scripts/cmake_options` to enable command line completion of the available cmake options. First source the convenience script (`simw-top/scripts/cmake_options`). In the fragment below the SW is built with the same SE05X_Auth option as above:

```
cd simw-top
source scripts/cmake_options.sh
cd simw-top_build/imx_native_se050_tloi2c
echo ${do # type double tab to enable command line completion hints
```

(continues on next page)

(continued from previous page)

```

${doApplet_A71CH_ON}                ${doHost_evkbimxrt1050_ON}
${doSE05X_Auth_AppletSCP03_ON}      ${doSE05X_Auth_None_ON}
${doApplet_A71CL_ON}                ${doHost_frdmk64f_ON}
...
...
${doHost_Win10IoT_ON}                ${doSCP_SCP03_SSS_ON}

cmake ${doSE05X_Auth_UserID_ON} .
cmake --build .

```

4.4.3 SSS Examples

Above build steps will also create a few examples that illustrate the usage of se05x. Location - `simw-top_build/imx_native_se050_tloi2c/bin`

Refer to *Demo and Examples* for details on these examples and for running cloud/tls demo applications

4.5 Raspberry Pi Build

4.5.1 Prerequisite

Linux should be running on the Raspberry Pi development board, the release was tested with Raspbian Buster (4.19.75-v7l+)

4.5.2 Build Instructions

- 1) Copy the Plug & Trust middleware package to the raspberry pi file system
- 2) Install required build tools, if the image does not have them already
- 3) Enable I2C if not yet enabled on your board. If `ls /sys/bus/i2c/devices` does not list `i2c-1`, I2C needs to be enabled for your board..
 - Run `sudo raspi-config`
 - Use the down arrow to select Interfacing Options.
 - Follow instructions and Enable I2C
- 4) Install extra build tools. e.g. on `2019-07-10-raspbian-buster-lite.img`, following packages are also required:

```
sudo apt-get install cmake cmake-curses-gui cmake-gui libssl-dev
```

- 5) Execute the below commands to build and install the se05x libraries to the system

```

cd simw-top
python scripts/create_cmake_projects.py
cd ../simw-top_build/raspberrypi_native_se050_tloi2c
cmake --build .
make install
ldconfig /usr/local/lib

```

Default cmake options are shown below:

Applet	SE050_C
CMAKE_BUILD_TYPE	Debug
CMAKE_INSTALL_PREFIX	/usr/local
Host	Raspbian
HostCrypto	OPENSSL
IOT	GCP
Log	Verbose
NXPInternal	ON
OpenSSL	1_1_1
RTOS	Default
SCP	None
SE05X_Auth	None
SMCOM	VCOM
SSS_HAVE_FIPS	0
WithCodeCoverage	OFF
WithNXPNFCRdLib	OFF
WithSSS_TestCounterPart	ON
WithSharedLIB	OFF
mbedtls_ALT	None

- 6) If required cmake options can be changes and libraries can be rebuilt and installed. Refer [CMake](#)

```
cd simw-top_build/raspberrypi_native_se050_tloi2c

# With Gui
cmake-gui .
```

The `cmake-gui` will bring up CMake GUI that can set up command with bring up the CMake UI using which options like logging level, etc. can be changed.

If you are connected to Raspberry PI over a command line terminal, (which does not have GUI), you can use `ccmake .` instead of `cmake-gui .`

4.5.3 SSS Examples

Above build steps will also build few sample examples for se05x test. Location: `simw-top_build/raspberrypi_native_se050_tloi2c/bin`

Refer [Demo and Examples](#) for details on these examples and for running cloud/tls demo applications

4.5.4 Enable Pin configuration for SE05X

Connect GPIO22 (P1_15) to enable pin of SE05X.

4.6 CMake

CMake is an open-source, cross-platform family of tools designed to build, test and package software. CMake is used to control the software compilation process using simple platform and compiler independent configuration files, and generate native makefiles and workspaces that can be used in the compiler environment of your choice. (From <https://cmake.org/>). CMake can be downloaded from <https://cmake.org/download/>.

4.6.1 Reference Commands

We recommend to use out of the source build of Cmake and run it from other directory.

A reference command to compiling for SE050_C from Windows PC is:

```
cd <ROOT_DIR>
mkdir ..\build_se050
cd ..\build_se050
cmake ..\<ROOT_DIR> -DApplet=SE050_C -DHost=PCWindows
```

A helper python script `scriptscreate_cmake_projects.py` is available as a part of the project.

Sample usage on Windows is:

```
call <ROOT_DIR>\scripts\env_setup.bat
python scripts\create_cmake_projects.py
```

Sample usage on Linux/POSIX is:

```
. <ROOT_DIR>/scripts/env_setup.sh
python scripts/create_cmake_projects.py
```

This will create a `<ROOT_DIR>_build` directory with some reference CMake projects.

Warning: It is recommend to keep `<ROOT_DIR>` to:

- A small path as much as possible. i.e. Closer to top most directory of the drive, instead of a deep long path.
- Is kept in a path that does not have spaces in it. This avoids issues faced with building some times.

Note: For sample paths of installation tools, please see `scriptsenv_setup.bat` and `scripts/env_setup.sh`

4.6.2 CMake - Cross compiling

CMake can also be used for cross compiling for non native platforms. For that tool-chain files have to be used. As part of the Plug & Trust Middleware the following files are used.

Android `$NDK_ROOT/build/cmake/android.toolchain.cmake`

iMX6 `scripts/ToolchainFile_imx6.cmake`

FRDM K64F `scripts/armgcc_force_cpp.cmake`

NDK_ROOT is the environment variable set by Android Build system.

4.7 CMake Options

4.7.1 Applet

Applet

The Secure Element Applet

You can compile host library for different Applets listed below. Please note, some of these Applets may be for NXP Internal use only.

`-DApplet=None`: Compiling without any Applet Support

`-DApplet=A71CH`: A71CH (ECC)

`-DApplet=SE05X_A`: SE050 Type A (ECC)

`-DApplet=SE05X_B`: SE050 Type B (RSA)

`-DApplet=SE05X_C`: SE050 (Super set of A + B)

4.7.2 SE05X_Ver

SE05X_Ver

SE50 Applet version.

03_XX would only enable features of version 03.XX version of applet. But, this would be compatibility would be added for newer versions of the Applet. When 04_XX is selected, it would expose features available in 04_XX at compile time.

`-DSE05X_Ver=03_XX`: SE050

4.7.3 Host

Host

Host where the software stack is running

e.g. Windows, PC Linux, Embedded Linux, Kinetis like embedded platform

`-DHost=Darwin`: OS X / Macintosh

`-DHost=PCLinux32`: PC/Laptop Linux with 32bit libraries

`-DHost=PCLinux64`: PC/Laptop Linux with 64bit libraries

`-DHost=PCWindows`: PC/Laptop Windows

- DHost=Cygwin: Using Cygwin
- DHost=frdmk64f: Embedded Kinetis Freedom K64F
- DHost=evkbimxrt1050: Embedded Kinetis i.MX RT 1050
- DHost=lpcpresso55s: Embedded LPCXpresso55s (No demarcation of secure/non-secure world)
- DHost=lpcpresso55s_ns: Non Secure world of LPCXpresso55s
- DHost=lpcpresso55s_s: Secure world of LPCXpresso55s
- DHost=iMXLinux: Embedded Linux on i.MX
- DHost=Raspbian: Embedded Linux on RaspBerry PI
- DHost=Android: Android

4.7.4 SMCOM

SMCOM

Communication Interface

How the host library communicates to the Secure Element. This may be directly over an I2C interface on embedded platform. Or sometimes over Remote protocol like JRCP_V1 / JRCP_V2 / VCOM from PC.

- DSMCOM=None: Not using any Communication layer
- DSMCOM=JRCP_V2: Socket Interface New Implementation
- DSMCOM=JRCP_V1: Socket Interface Old Implementation.** This is the interface used from Host PC when we run jrctp1_server from the linux PC.
- DSMCOM=VCOM: Virtual COM Port
- DSMCOM=SCI2C: Smart Card I2C for A71CH and A71CH
- DSMCOM=T1oI2C: T=1 over I2C for SE050
- DSMCOM=PCSC: CCID PC/SC reader interface

4.7.5 HostCrypto

HostCrypto

Counterpart Crypto on Host

What is being used as a cryptographic library on the host. As of now only OpenSSL / mbedTLS is supported

- DHostCrypto=MBEDTLS: Use mbedTLS as host crypto
- DHostCrypto=OPENSSL: Use OpenSSL as host crypto
- DHostCrypto=User: User Implementation of Host Crypto** e.g. Files at `sss/src/user/crypto` have low level AES/CMAC primitives. The files at `sss/src/user` use those primitives. This becomes an example for users with their own AES Implementation This then becomes integration without mbedTLS/OpenSSL for SCP03 / AESKey.

Note: ECKey abstraction is not implemented/available yet.

-DHostCrypto=None: NO Host Crypto Note, this is unsecure and only provided for experimentation on platforms that do not have an mbedTLS PORT Many *Feature Control* have to be disabled to have a valid build.

4.7.6 RTOS

RTOS

Choice of Operating system

Default would mean nothing special. i.e. Without any RTOS on embedded system, or default APIs on PC/Linux

-DRTOS=Default: No specific RTOS. Either bare metal on embedded system or native linux or Windows OS

-DRTOS=FreeRTOS: Free RTOS for embedded systems

4.7.7 mbedTLS_ALT

mbedTLS_ALT

ALT Engine implementation for mbedTLS

When set to None, mbedTLS would not use ALT Implementation to connect to / use Secure Element. This needs to be set to SSS for Cloud Demos over SSS APIs

-DmbedTLS_ALT=SSS: Use SSS Layer ALT implementation

-DmbedTLS_ALT=A71CH: Legacy implementation

-DmbedTLS_ALT=None: Not using any mbedTLS_ALT

When this is selected, cloud demos can not work with mbedTLS

4.7.8 SCP

SCP

Secure Channel Protocol

In case we enable secure channel to Secure Element, which interface to be used.

-DSCP=None

-DSCP=SCP03_SSS: Use SSS Layer for SCP. Used for SE050 family.

-DSCP=SCP03_HostCrypto: Use Host Crypto Layer for SCP03. Legacy implementation. Used for older demos of A71CH Family.

4.7.9 SE05X_Auth

SE05X_Auth

SE050 Authentication

This settings is used by examples to connect using various options to authenticate to the Applet.

-DSE05X_Auth=None: Use the default session (i.e. session less) login

-DSE05X_Auth=UserID: Do User Authentication with UserID

-DSE05X_Auth=PlatfSCP03: Use Platform SCP for connection to SE

-DSE05X_Auth=AESKey: Do User Authentication with AES Key Earlier this was called AppletSCP03

-DSE05X_Auth=ECKey: Do User Authentication with EC Key Earlier this was called FastSCP

-DSE05X_Auth=UserID_PlatfSCP03: UserID and PlatfSCP03

-DSE05X_Auth=AESKey_PlatfSCP03: AESKey and PlatfSCP03

-DSE05X_Auth=ECKey_PlatfSCP03: ECKey and PlatfSCP03

4.7.10 A71CH_AUTH

A71CH_AUTH

A71CH Authentication

This settings is used by SSS-API based examples to connect using either plain or authenticated to the A71CH.

-DA71CH_AUTH=None: Plain communication, not authenticated or encrypted

-DA71CH_AUTH=SCP03: SCP03 enabled

4.7.11 Log

Log

Logging

-DLog=Default: Default Logging

-DLog=Verbose: Very Verbose logging

-DLog=Silent: Totally silent logging

4.7.12 CMAKE_BUILD_TYPE

CMAKE_BUILD_TYPE

See https://cmake.org/cmake/help/latest/variable/CMAKE_BUILD_TYPE.html

For embedded builds, this choices sets optimization levels. For MSVC builds, build type is selected from IDE
As well

-DCMAKE_BUILD_TYPE=Debug: For developer

-DCMAKE_BUILD_TYPE=Release: Optimization enabled and debug symbols removed

-DCMAKE_BUILD_TYPE=RelWithDebInfo: Optimization enabled but with debug symbols

-DCMAKE_BUILD_TYPE=: Empty Allowed

4.7.13 Feature Control

Using these options, you can enable/disable individual features.

SSSFTR_SE05X_AES

SE05X Secure Element : Symmetric AES

SSSFTR_SE05X_ECC

SE05X Secure Element : Elliptic Curve Cryptography

SSSFTR_SE05X_RSA

SE05X Secure Element : RSA

SSSFTR_SE05X_KEY_SET

SE05X Secure Element : KEY operations : SET Key

SSSFTR_SE05X_KEY_GET

SE05X Secure Element : KEY operations : GET Key

SSSFTR_SE05X_AuthEckKey

SE05X Secure Element : Authenticate via ECKKey

SSSFTR_SE05X_AuthSession

SE05X Secure Element : Allow creation of user/authenticated session.

If the intended deployment only uses Platform SCP Or it is a pure session less integration, this can save some code size.

SSSFTR_SE05X_CREATE_DELETE_CRYPTOBJ

SE05X Secure Element : Allow creation/deletion of Crypto Objects

If disabled, new Crypto Objects are neither created and old/existing Crypto Objects are not deleted. It is assumed that during provisioning phase, the required Crypto Objects are pre-created or they are never going to be needed.

SSSFTR_SW_AES

Software : Symmetric AES

SSSFTR_SW_ECC

Software : Elliptic Curve Cryptography

SSSFTR_SW_RSA

Software : RSA

SSSFTR_SW_KEY_SET

Software : KEY operations : SET Key

SSSFTR_SW_KEY_GET

Software : KEY operations : GET Key

SSSFTR_SW_TESTCOUNTERPART

Software : Used as a test counterpart

e.g. Major part of the mebdTLS SSS layer is purely used for testing of Secure Element implementation, and can be avoided fully during many production scenarios.

4.7.14 Deprecated Defines

Keep and for time being for backwards compatibility. They will be removed in some future release.

- WithApplet_SE05X is renamed to SSS_HAVE_APPLET_SE05X_IOT
- WithApplet_SE050_A is renamed to SSS_HAVE_APPLET_SE05X_A
- WithApplet_SE050_B is renamed to SSS_HAVE_APPLET_SE05X_B
- WithApplet_SE050_C is renamed to SSS_HAVE_APPLET_SE05X_C
- SSS_HAVE_SE050_A is renamed to SSS_HAVE_APPLET_SE05X_A
- SSS_HAVE_SE050_B is renamed to SSS_HAVE_APPLET_SE05X_B
- SSS_HAVE_SE050_C is renamed to SSS_HAVE_APPLET_SE05X_C
- SSS_HAVE_SE05X is renamed to SSS_HAVE_APPLET_SE05X_IOT

- SSS_HAVE_SE is renamed to SSS_HAVE_APPLET
- SSS_HAVE_LOOPBACK is renamed to SSS_HAVE_APPLET_LOOPBACK
- SSS_HAVE_ALT is renamed to SSS_HAVE_MBEDTLS_ALT
- WithApplet_None is renamed to SSS_HAVE_APPLET_NONE
- SSS_HAVE_None is renamed to SSS_HAVE_APPLET_NONE
- WithApplet_A71CH is renamed to SSS_HAVE_APPLET_A71CH
- SSS_HAVE_A71CH is renamed to SSS_HAVE_APPLET_A71CH
- WithApplet_A71CL is renamed to SSS_HAVE_APPLET_A71CL
- SSS_HAVE_A71CL is renamed to SSS_HAVE_APPLET_A71CL
- WithApplet_A71CH_SIM is renamed to SSS_HAVE_APPLET_A71CH_SIM
- SSS_HAVE_A71CH_SIM is renamed to SSS_HAVE_APPLET_A71CH_SIM
- WithApplet_SE05X_A is renamed to SSS_HAVE_APPLET_SE05X_A
- SSS_HAVE_SE05X_A is renamed to SSS_HAVE_APPLET_SE05X_A
- WithApplet_SE05X_B is renamed to SSS_HAVE_APPLET_SE05X_B
- SSS_HAVE_SE05X_B is renamed to SSS_HAVE_APPLET_SE05X_B
- WithApplet_SE05X_C is renamed to SSS_HAVE_APPLET_SE05X_C
- SSS_HAVE_SE05X_C is renamed to SSS_HAVE_APPLET_SE05X_C
- WithApplet_SE05X_L is renamed to SSS_HAVE_APPLET_SE05X_L
- SSS_HAVE_SE05X_L is renamed to SSS_HAVE_APPLET_SE05X_L
- WithApplet_LoopBack is renamed to SSS_HAVE_APPLET_LOOPBACK
- SSS_HAVE_LoopBack is renamed to SSS_HAVE_APPLET_LOOPBACK
- SSS_HAVE_MBEDTLS is renamed to SSS_HAVE_HOSTCRYPTO_MBEDTLS
- SSS_HAVE_MBEDCRYPTO is renamed to SSS_HAVE_HOSTCRYPTO_MBEDCRYPTO
- SSS_HAVE_OPENSSL is renamed to SSS_HAVE_HOSTCRYPTO_OPENSSL
- SSS_HAVE_USER is renamed to SSS_HAVE_HOSTCRYPTO_USER
- SSS_HAVE_NONE is renamed to SSS_HAVE_HOSTCRYPTO_NONE
- SSS_HAVE_ALT_SSS is renamed to SSS_HAVE_MBEDTLS_ALT_SSS
- SSS_HAVE_ALT_A71CH is renamed to SSS_HAVE_MBEDTLS_ALT_A71CH
- SSS_HAVE_ALT_NONE is renamed to SSS_HAVE_MBEDTLS_ALT_NONE
- SSS_HAVE_SE05X_Auth_None is renamed to SSS_HAVE_SE05X_AUTH_NONE
- SSS_HAVE_SE05X_Auth_UserID is renamed to SSS_HAVE_SE05X_AUTH_USERID
- SSS_HAVE_SE05X_Auth_PlatformSCP03 is renamed to SSS_HAVE_SE05X_AUTH_PLATFSCP03
- SSS_HAVE_SE05X_Auth_AESKey is renamed to SSS_HAVE_SE05X_AUTH_AESKEY
- SSS_HAVE_SE05X_Auth_ECKey is renamed to SSS_HAVE_SE05X_AUTH_ECKEY
- SSS_HAVE_SE05X_Auth_UserID_PlatformSCP03 is renamed to SSS_HAVE_SE05X_AUTH_USERID_PLATFSCP03
- SSS_HAVE_SE05X_Auth_AESKey_PlatformSCP03 is renamed to SSS_HAVE_SE05X_AUTH_AESKEY_PLATFSCP03

- `SSS_HAVE_SE05X_Auth_ECKey_PlatformSCP03` is renamed to `SSS_HAVE_SE05X_AUTH_ECKEY_PLATFSCP03`

WithNXPNFCRdLib

- Compile in NXP NFC RdLib support
- Default is OFF
- Use NXP NFC RdLib. This is used mainly for RC663 + SAM Use Cases. Package available under NDA is needed to use this feature

WithOPCUA_open62541

- Compile With open62541 Support
- Default is OFF
- Compile with OPC UA. By default it is disabled from compilation.

WithSharedLIB

- Create and use shared libraries
- Default is OFF
- Create shared libraries. Applicable for Engine DLL and other use cases.

4.7.15 NXP Internal Options

These options are not supported outside NXP.

NXPInternal

- NXP Internal
- Default is OFF. (ON only within NXP)

Note: For deliveries outside NXP, this option is disabled.

WithCodeCoverage

- Compile with Code Coverage
- Default is OFF

4.7.16 Other Variables

WithExtCustomerCode

- Include code from `../customer`
- Default is OFF
- Include code from external folder. This way, experimental code can be included in build from outside the `simw-top` repository.

SIMW_INSTALL_INC_DIR

- Location where library header files are installed for linux based targets. (Used for iMX Linux)
- Default location is `</usr/local/>include/se05x`

SIMW_INSTALL_SHARE_DIR

- Location where miscellaneous scripts get copied for linux based targets. (Used for iMX Linux)
- e.g. `cmake_options.mak` which has current cmake build settings.
- Default location is `</usr/local/>share/se05x`

DEMO AND EXAMPLES

Some of the examples below can be run on windows or raspberry-pi with optional connection string (VCOM or I2C address) as command line argument. When running examples on raspberry pi connection string of secured element should be passed in the format <i2c_port>:<i2c_addr>

Example Raspberry Pi:

```
./ex_ecc "/dev/i2c-1:0x48"
```

On Windows with VCOM:

```
ex_ecc.exe COM1
```

5.1 DEMO List

5.1.1 Platforms List

KSDK Embedded platforms like FRDM K64F, i.MX RT1050, LPC55S

KSDK-CLOUD Embedded platforms like FRDM K64F, i.MX RT1050, LPC55S, that can connect to cloud.

LINUX Linux based platforms/systems like iMX6, iMX8, Raspberry Pi

PC Windows PC

ALL KSDK, LINUX, PC

5.1.2 SSS APIs Examples

Demo	Description	Platforms supported	SE supported
Section 5.2.1 <i>ECC Example</i>	Inject ECC Key and use it for sign and verify operation	ALL	SE05X (A and C), A71CH
Section 5.2.2 <i>RSA Example</i>	Generate RSA key and use it for sign and verify operation	ALL	SE05X (B and C)
Section 5.2.3 <i>Symmetric AES Example</i>	Inject AES key, encrypt and decrypt data with it	ALL	SE05X
Section 5.2.4 <i>HKDF Example</i>	HMAC Key derivation operation based on the info and salt. Inject HMAC key into SE and derive a key using HMAC from the SE into the host keystore	ALL	SE05X, A71CH
Section 5.2.5 <i>Message Digest Example</i>	Message Digest hashing operation. Calculate SHA256 over data.	ALL	SE05X, A71CH
Section 5.2.6 <i>HMAC Example</i>	Inject HMAC key and calculate a HMAC	ALL	SE05X, A71CH
Section 5.2.7 <i>ECDH Example</i>	inject ECC key into SE and derive a key using ECDH from the SE into the host keystore.	ALL	SE05X, A71CH

5.1.3 Cloud connectivity Examples

Demo	Description	Platforms supported	SE supported
Section 5.3 <i>AWS Demo for KSDK</i>	Connect to Amazon Web Services IoT Core	KSDK-CLOUD	SE05X, A71CH
Section 5.4 <i>AWS Demo for iMX Linux / RaspberryPi</i>	Connect to Amazon Web Services	LINUX	SE05X, A71CH
Section 5.5 <i>GCP Demo for KSDK</i>	Connect to Google Cloud	KSDK-CLOUD	SE05X, A71CH
Section 5.6 <i>GCP Demo for iMX Linux / Raspberry Pi</i>	Connect to Google Cloud	LINUX	SE05X, A71CH
Section 5.7 <i>IBM Watson Demo for KSDK</i>	Connect to IBM Watson	KSDK-CLOUD	SE05X, A71CH
Section 5.8 <i>IBM Watson Demo for iMX Linux / Raspberry Pi</i>	Connect to IBM Watson	LINUX	SE05X, A71CH
Section 5.9 <i>Azure Demo for KSDK</i>	Connect to Microsoft Azure	KSDK-CLOUD	SE05X, A71CH
Section 5.10 <i>Azure Demo for iMX Linux / Raspberry Pi</i>	Connect to Microsoft Azure	LINUX	SE05X, A71CH
Section 5.11 <i>Greengrass Demo for Linux</i>	Connect as AWS Greengrass Core	Raspberry PI	SE05X

5.1.4 OpenSSL Engine Examples

Demo	Description	Platforms supported	SE supported
Section 5.12 <i>OpenSSL Engine: TLS Client example for iMX/Rpi3</i>	Setting up a TLS Link using OpenSSL Engine	LINUX	SE05X, A71CH

5.1.5 mbedTLS Examples

Demos regarding the mbedTLS ALT implementation. See *Introduction on mbedTLS ALT Implementation*

Demo	Description	Platforms supported	SE supported
SSL2 Client	Use extended SSL Client 2 & SSL Server 2 from mbedTLS	PC	SE05X, A71CH
DTLS Client	Use extended dtls_client & dtls_server from mbedTLS	PC	SE05X, A71CH

5.1.6 OPC UA Examples

Demo	Description	Platforms supported	SE supported
Section 5.13 <i>OPC UA (Open62541) Demo</i>	OPC UA Server	PC, iMX6	SE05X

5.1.7 SE05X Specific Examples

Demo	Description	Platforms supported
Section 5.14 <i>SE05X Minimal example</i>	Showcase usage of SE05X low level APIs	ALL
Section 5.15 <i>SE05X Get Info example</i>	Showcase Platform details of SE05X	ALL
Section 5.16 <i>APDU Player Demo</i>	Send RAW APDUs to SE050	PC, LINUX
Section 5.17 <i>Using policies for secure objects</i>	Showcase usage of policies	ALL
Section 5.18 <i>Get Certificate from the SE</i>	Read the certificate from the SE and store it on the file system.	ALL (With mbedTLS Only)
Section 5.19 <i>SE05X Rotate PlatformSCP Keys Demo</i>	Showcase Rotation of SE05X PlatformSCP03 Keys	ALL
Section 5.20 <i>I2C Master Example</i>	Showcase usage of I2CM interface of SE050	ALL
Section 5.21 <i>SE05X WiFi KDF Example</i>	Showcase usage of PBKDF	ALL
Section 5.23 <i>SE05X Export Transient objects</i>	Export transient objects	PC, LINUX
Section 5.22 <i>SE05X Import Transient objects</i>	Import transient objects	PC, LINUX
Section 5.24 <i>Import External Object Prepare</i>	Create ImportExternlObject raw APDU	PC, LINUX
Section 5.25 <i>Import External Object Create</i>	Import external objects securely	PC, LINUX
Section 5.28 <i>SE05X Transport Lock example</i>	Show transport lock feature	PC, LINUX
Section 5.29 <i>SE05X Transport UnLock example</i>	Show transport unlock feature	PC, LINUX
Section 5.27 <i>Read object with Attestation</i>	Demonstrate how to read object with attestation	ALL
Section 5.30 <i>SE05X Timestamp</i>	Demonstrate increment of timestamp inside SE	ALL
Section 5.38 <i>Write APDU to buffer</i>	Demonstrate how to write APDU to buffer	ALL

5.1.8 NFC (DESFire) Examples

Demos that interact with DESFire card via RC663. These examples can be run from:

- From **KSDK** with RC663
- From PC with FRDM-K64F & RC663

Warning: To run this example, you would need the nxpnfcrdlib component for which a Non-Disclosure Agreement(NDA) needs to be signed. Please contact your FAE for additional details.

Demo	Description
Section 5.31 <i>MIFARE DESFire EV2 : Prepare SE050</i>	Prepare/Provision SE050 with reference Keys. This example does not use RC663
Section 5.32 <i>MIFARE DESFire EV2 : Prepare MFD FEV2</i>	Prepare/Provision DESFireEv2 with reference Keys. This example does not use SE050.
Section 5.33 <i>MIFARE DESFire EV2 : Authentication</i>	Authenticate MIFARE DESFire EV2 using SE050 & RC663
Section 5.34 <i>MIFARE DESFire EV2 : Change Key</i>	MIFARE DESFire EV2 Change Key using SE050 & RC663
Section 5.35 <i>MIFARE DESFire EV2 : Diversified Change Key</i>	MIFARE DESFire EV2 Diversified Change Key using SE050 & RC663

5.1.9 Examples that use OpenSSL

Demo	Description	Platforms supported	SE supported
Section 5.36 <i>Tool to create Reference key file</i>	Native example to generate refKeys. (Only for NIST-P256 curve).	LINUX, PC	SE05X, A71CH
Section 5.37 <i>Building a self-signed certificate</i>	Create self signed certificates	LINUX, PC	SE05X

5.1.10 Ease of Use Configuration Examples

Seps for using the Ease Of Use Configuration of SE050.

Demo
Section 5.39 <i>Ease of Use configuration - IBM Watson</i>
Section 5.40 <i>Ease of Use configuration - Google Cloud Platform</i>
Section 5.41 <i>Ease of Use configuration - Azure IoT Hub</i>
Section 5.42 <i>Ease of Use configuration - AWS IoT Console</i>

5.2 SSS API Examples

5.2.1 ECC Example

This project demonstrates Elliptic Curve Cryptography sign and verify operation using SSS APIs

Prerequisites

- Build Plug & Trust middleware stack. (Refer *Building / Compiling*)

About the Example

This example does a elliptic curve cryptography signing and verify operation.

It uses the following APIs and data types:

- `sss_asymmetric_context_init()`
- `kAlgorithm_SSS_SHA256` from `sss_algorithm_t`
- `kMode_SSS_Sign` from `sss_mode_t`
- `sss_asymmetric_sign_digest()`
- `kMode_SSS_Verify` from `sss_mode_t`
- `sss_asymmetric_verify_digest()`

Console output

If everything is successful, the output will be similar to:

```
App :INFO :Running Elliptic Curve Cryptography Example ex_sss_ecc.c
App :INFO :Do Signing
App :INFO :digest (Len=32)
48 65 6C 6C 6F 20 57 6F 72 6C 64 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
App :INFO :signature (Len=72)
30 46 02 21 00 C8 4C 40 74 35 42 D7 37 64 03 D9
B1 1B 9C 0B 44 50 DC 70 1E 92 07 92 78 BC 0E C5
A4 07 FC 95 09 02 21 00 CA 70 02 36 13 65 47 72
0F 60 78 59 EA 59 81 82 DF 80 FD 89 2D FC 3E 7D
B2 FC 51 17 30 9B C4 15
App :INFO :Signing Successful !!!
App :INFO :Do Verify
App :INFO :digest (Len=32)
48 65 6C 6C 6F 20 57 6F 72 6C 64 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
App :INFO :signature (Len=72)
30 46 02 21 00 C8 4C 40 74 35 42 D7 37 64 03 D9
B1 1B 9C 0B 44 50 DC 70 1E 92 07 92 78 BC 0E C5
A4 07 FC 95 09 02 21 00 CA 70 02 36 13 65 47 72
0F 60 78 59 EA 59 81 82 DF 80 FD 89 2D FC 3E 7D
B2 FC 51 17 30 9B C4 15
App :INFO :Verification Successful !!!
```

(continues on next page)

(continued from previous page)

```
App :INFO :ex_sss_ecc Example Success !!!...
App :INFO :ex_sss Finished
```

5.2.2 RSA Example

This project demonstrates RSA sign and verify operations using SSS APIs.

Prerequisites

- Build Plug & Trust middleware stack. (Refer *Building / Compiling*)

About the Example

This example does a RSA signing and verify operation.

It uses the following APIs and data types:

- `sss_asymmetric_context_init()`
- `kAlgorithm_SSS_RSASSA_PKCS1_PSS_MGF1_SHA256` from `sss_algorithm_t`
- `kMode_SSS_Sign` from `sss_mode_t`
- `kSSS_CipherType_RSA` from `:cpp:enumerator: sss_cipher_type_t`
- `sss_asymmetric_sign_digest()`
- `kMode_SSS_Verify` from `sss_mode_t`
- `sss_asymmetric_verify_digest()`

Note: This example tries to delete key first. Deletion would be successful, if the key already exists. Otherwise it would return an error message which is perfectly alright and the example could be successfully executed.

Console output

If everything is successful, the output will be similar to:

```
App :INFO :Running RSA Example ex_sss_rsa.c
sss :WARN :nxEnsure:'ret == SM_OK' failed. At Line:5612 Function:sss_se05x_TXn
sss :WARN :Could not delete Key id EF000046
App :INFO :Delete key succeeds only if key exists, ignore error message if any
App :INFO :Do Signing
App :INFO :digest (Len=32)
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
App :INFO :Signing successful !!!
App :INFO :signature (Len=512)
60 64 F1 3B DE 72 F7 91 05 29 A4 1D AC 48 47 D6
25 E6 D4 35 67 32 78 37 45 90 03 56 1E B4 97 F5
2A 09 26 CE 9A 81 4E 24 07 31 1E B4 18 04 BD B8
88 42 6F 95 C6 3A 8A 6F 09 7B 80 35 4A F6 1D 9F
```

(continues on next page)

(continued from previous page)

```

19 6F 99 66      F9 61 47 6E      D0 52 8F 80      C5 E4 D3 F6
A2 37 B0 3A      8B 34 32 A0      00 2D 0D B7      BA 10 19 C9
58 EB 7D D0      7E E9 47 4E      E1 E7 2C 96      44 0E D5 BF
4A F4 16 4A      8E BB C0 1C      8F A5 AD C4      64 3A 13 89
B0 08 03 00      19 1C BD 7C      E1 4A 69 AC      C7 1D B2 A9
DB 80 68 34      EA B9 96 1C      D8 DE F0 E2      09 CA A0 4D
35 38 6B FC      88 A3 A2 B8      E9 15 18 7B      5D 81 B4 C6
53 2D 92 12      5E 7D 84 D1      7A 2E C1 C4      72 6D 5F 29
95 4E 21 EE      47 AC 29 80      4E D4 03 DE      98 1B 6C 82
55 D2 61 47      29 CD D1 AE      B8 C4 89 85      89 FB 3E BE
20 30 BE A7      AB DA 5B 42      1C 10 F8 3B      5E 8A C9 23
4F DE 8F 3A      01 D1 ED 4C      79 CF D8 2A      47 4D FD 7E
12 20 D4 B1      49 4A 27 A9      3A 72 34 B7      35 39 4F 87
0D C8 C2 D1      91 E7 93 E5      54 D3 1B 0D      D6 30 97 CD
56 33 9F 54      03 43 E9 44      0A 22 4D D2      27 5C 42 BD
82 40 F7 D8      83 3E 10 A8      31 43 A2 0D      8A 1F 27 FE
66 7A 12 63      2F DC F5 9F      6C 83 B1 C3      AC F1 A9 2F
96 EE 94 00      5C 57 9D AB      D3 3F A0 EB      F0 6B E1 33
95 AB 14 5D      07 87 C0 14      18 70 51 A2      EF 0E 1B C4
92 07 CC 18      3F 0F 48 80      FA 85 55 BD      B9 86 F3 C2
DC 61 84 A6      84 19 4E 9D      60 99 2A A9      84 43 38 42
08 61 5E 53      43 06 C4 BB      72 42 8D 41      A3 3D 64 3D
22 2B 11 1D      13 88 1D CF      03 9C C2 C7      71 DB 4A FA
58 B1 AF 13      8C 86 55 16      3E 15 BB EF      CD 2F EC CD
A8 CA F7 7C      E9 B4 1B ED      FE A1 89 A3      AF 03 7B 38
2F B5 AE 30      0F 88 41 D4      2A E1 F8 4E      25 88 4C B2
2B FD 6A 98      60 B9 F8 A9      A3 5E A2 68      A1 C9 11 F1
BB 06 67 2F      8E B0 5A A6      55 C7 1E 3A      FC 71 CC 1A
App :INFO :Do Verification
App :INFO :Verification successful !!!
App :INFO :ex_sss_RSA Example Success !!!...
App :INFO :ex_sss Finished

```

5.2.3 Symmetric AES Example

This project demonstrates symmetric cryptography - AES encryption and decryption operations.

Prerequisites

- Build Plug & Trust middleware stack. (Refer *Building / Compiling*)

About the Example

This example does a symmetric cryptography AES encryption and decryption operation.

It uses the following APIs and data types:

- `sss_symmetric_context_init()`
- `kAlgorithm_SSS_AES_CBC` from `sss_algorithm_t`
- `kSSS_CipherType_AES` from `sss_cipher_type_t`
- `kMode_SSS_Encrypt` from `sss_mode_t`
- `sss_cipher_one_go()`

- `kMode_SSS_Decrypt` from `sss_mode_t`

Console output

If everything is successful, the output will be similar to:

```
App :INFO :Running AES symmetric Example ex_sss_symmetric.c
App :INFO :Do Encryption
App :INFO :iv (Len=16)
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
App :INFO :srcData (Len=16)
48 45 4C 4C 4F 48 45 4C 4C 4F 48 45 4C 4C 4F 31
App :INFO :Encryption successful !!!
App :INFO :encrypted data (Len=16)
32 A6 04 88 C5 B3 FF 40 50 AF 56 A5 68 AE D1 05
App :INFO :Do Decryption
App :INFO :iv (Len=16)
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
App :INFO :Encrypted data (Len=16)
32 A6 04 88 C5 B3 FF 40 50 AF 56 A5 68 AE D1 05
App :INFO :Decryption successful !!!
App :INFO :decrypted data (Len=16)
48 45 4C 4C 4F 48 45 4C 4C 4F 48 45 4C 4C 4F 31
App :INFO :ex_sss_symmetric Example Success !!!...
App :INFO :ex_sss Finished
```

5.2.4 HKDF Example

This project demonstrates an HMAC Key derivation operation based on info and salt using SSS APIs.

Prerequisites

- Build Plug & Trust middleware stack. (Refer [Building / Compiling](#))

About the Example

This example does a HMAC Key derivation operation based on the info and salt.

It uses the following APIs and data types:

- `sss_derive_key_context_init()`
- `kAlgorithm_SSS_HMAC_SHA256` from `sss_algorithm_t`
- `kMode_SSS_ComputeSharedSecret` from `sss_mode_t`
- `kSSS_CipherType_HMAC` from `sss_cipher_type_t`
- `sss_derive_key_go()`

Console output

If everything is successful, the output will be similar to:

```
App :INFO :Running HMAC Key Derivation Function Example ex_sss_hkdf.c
App :INFO :Do Key Derivation
App :INFO :salt (Len=32)
    AA 1A 2A E3    B2 76 15 4D    67 F9 D8 4C    B9 35 54 56
    BB 1B 2B 03    04 05 06 07    08 09 0A 0B    0C 0D 0E 0F
App :INFO :info (Len=192)
    00 01 02 03    04 05 06 07    08 09 0A 0B    0C 0D 0E 0F
    10 11 12 13    14 15 16 17    18 19 1A 1B    1C 1D 1E 1F
    20 21 22 23    24 25 26 27    28 29 2A 2B    2C 2D 2E 2F
    30 31 32 33    34 35 36 37    38 39 3A 3B    3C 3D 3E 3F
    40 41 42 43    44 45 46 47    48 49 4A 4B    4C 4D 4E 4F
    50 51 52 53    54 55 56 57    58 59 5A 5B    5C 5D 5E 5F
    60 61 62 63    64 65 66 67    68 69 6A 6B    6C 6D 6E 6F
    70 71 72 73    74 75 76 77    78 79 7A 7B    7C 7D 7E 7F
    80 81 82 83    84 85 86 87    88 89 8A 8B    8C 8D 8E 8F
    90 91 92 93    94 95 96 97    98 99 9A 9B    9C 9D 9E 9F
    A0 A1 A2 A3    A4 A5 A6 A7    A8 A9 AA AB    AC AD AE AF
    B0 B1 B2 B3    B4 B5 B6 B7    B8 B9 BA BB    BC BD BE BF
App :INFO : Key Derivation successful !!!
App :INFO :hkdfOutput (Len=128)
    6E 69 25 DB    1D D7 37 64    3C 9F F5 02    0D 54 B1 0A
    97 FF 80 78    36 A6 86 80    6D B5 77 5C    89 DD 61 53
    E3 69 58 67    6B 41 EB 4A    8B 10 01 F6    BB 67 7C A4
    26 F7 87 DE    03 A1 18 B1    D3 ED 11 38    CA 0A AA 02
    0A C2 A0 B1    65 7D 80 83    23 7C 8B EA    58 EE E4 DF
    DF 17 49 59    8B 66 62 7F    C7 EE 63 87    5A 1A F4 C3
    BB 97 3F 35    47 06 4A EA    15 DA 15 21    5B 87 DA AA
    81 1F 1F B0    D7 95 4E F4    25 5D C6 75    34 2B 0C 40
App :INFO :ex_sss_hkdf Example Success !!!!...
App :INFO :ex_sss Finished
```

5.2.5 Message Digest Example

This project demonstrates a Message Digest / hashing operation using SSS APIs.

Prerequisites

- Build Plug & Trust middleware stack. (Refer [Building / Compiling](#))

About the Example

This example does a Message Digest hashing operation.

It uses the following APIs and data types:

- `sss_digest_context_init()`
- `kAlgorithm_SSS_SHA1` from `sss_algorithm_t`
- `kMode_SSS_Digest` from `sss_mode_t`
- `sss_digest_one_go()`

Console output

If everything is successful, the output will be similar to:

```
App :INFO :Running Message Digest Example ex_sss_md.c
App :INFO :Do Digest
App :INFO :input (Len=10)
48 65 6C 6C 6F 57 6F 72 6C 64
App :INFO :Message Digest successful !!!
App :INFO :digest (Len=32)
87 2E 4E 50 CE 99 90 D8 B0 41 33 0C 47 C9 DD D1
1B EC 6B 50 3A E9 38 6A 99 DA 85 84 E9 BB 12 C4
App :INFO :ex_sss_digest Example Success !!!!...
App :INFO :ex_sss Finished
```

5.2.6 HMAC Example

This project demonstrates a HMAC operation on a message using SSS APIs.

Prerequisites

- Build Plug & Trust middleware stack. (Refer *Building / Compiling*)

About the Example

This example does a HMAC operation on input data.

It uses the following APIs and data types:

- `sss_mac_context_init()`
- `kAlgorithm_SSS_HMAC_SHA256` from `sss_algorithm_t`
- `kMode_SSS_Mac` from `sss_mode_t`
- `sss_mac_one_go()`

Console output

If everything is successful, the output will be similar to:

```
App :INFO :Running HMAC (SHA256) Example ex_sss_hmac.c
App :INFO :Do HMAC
App :INFO :input (Len=10)
48 65 6C 6C 6F 57 6F 72 6C 64
App :INFO :hmac key (Len=16)
48 65 6C 6C 6F 48 65 6C 6C 6F 48 65 6C 6C 6F 48
App :INFO :HMAC (SHA256) successful !!!
App :INFO :hmac (Len=32)
68 7A 26 95 49 67 9D 6E FA 11 19 5E 96 CB BA C2
6B 50 A5 09 10 8A D1 48 B5 FC A0 94 2C BD 10 21
App :INFO :ex_sss_hmac Example Success !!!!...
App :INFO :ex_sss Finished
```

5.2.7 ECDH Example

This project demonstrates generating a ECDH key using SSS APIs.

Prerequisites

- Build Plug & Trust middleware stack. (Refer *Building / Compiling*)

About the Example

This example generates a ECDH key.

It uses the following APIs and data types:

- `sss_derive_key_context_init()`
- `kAlgorithm_SSS_ECDH` from `sss_algorithm_t`
- `kMode_SSS_ComputeSharedSecret` from `sss_mode_t`
- `sss_derive_key_dh()`

Console output

If everything is successful, the output will be similar to:

```
App :INFO :Running ECDH Example ex_sss_ecdh.c
App :INFO :ECDH successful !!!
App :INFO :ECDH derive Key (Len=32)
      C2 EA 1C 13      7D 30 F7 DA      65 1E B3 DB      7A F1 CF 42
      DF 38 B2 E6      22 41 2B 5B      BB DC F5 10      8E D5 69 CB
App :INFO :ex_sss_ecdh Example Success !!!!...
App :INFO :ex_sss Finished
```

5.3 AWS Demo for KSDK

This demo demonstrates connection to AWS IoT Console using pre-provisioned device credentials and publish/subscribe procedure using MQTT.

5.3.1 Prerequisites

- Active AWS account
- MCUXpresso installed (for running aws demo on k64)
- Any Serial communicator
- Flash VCOM binary on the device. VCOM binary can found under <PROJECT>binaries folder.
- Refer to *CLI Tool* for pyCLI tool setup

5.3.2 Using WiFi with LPC55S

WiFi shield Silex-2401 is supported with LPC55S. Mount the WiFi shield on to the arduino stackable headers.

5.3.3 Creating a device on AWS account

TO BE UPDATED,

5.3.4 Creating and updating device keys and certificates to SE

- 1) Complete *Section 7.3 Steps needed before running ssscli tool*
- 2) Check the vcom port number
- 3) To create certificates on windows and provision, go to `simw-top/pycli` directory and call:

```
call venv\Scripts\activate.bat
cd Provisioning
python GenerateAWSCredentials.py <COM_PORT>
python ResetAndUpdate_AWS.py <COM_PORT>
```

- 4) Certificates and Keys are generated at `simw-top/pycli/Provisioning/aws`

5.3.5 Running the Demo

- 1) Open `frdmk64f_mbedtls_aws_demo` project found under `<PROJECT>projects` in MCUXPRESSO IDE
- 2) Open a serial terminal on PC for OpenSDA serial device with these settings:

```
- 115200 baud rate
- 8 data bits
- No parity
- One stop bit
- No flow control
- change Setup->Terminal->New-line->Receive->AUTO
```

- 3) Connect the boards's RJ45 to network with Internet access (IP address to the board is assigned by the DHCP server). Make sure the connection on port 8883 is not blocked.
- 4) Download the program to the target board.
- 5) Either press the reset button on your board or launch the debugger in your IDE to begin running the demo.
- 6) The BLUE LED is turned ON during boot
- 7) Persistent RED LED ON indicates error
- 8) **First time during connection, the device certificate needs to be**
 - Activated
 - Attached with a policy that allows usage of this certificate
- 9) All lights off along with the following message indicates readiness to subscribe messages from AWS:


```
Subscribing...
-->sleep
-->sleep
Publish done
```

In AWS IOT shadow, the following indicates the state of the LED:

```
{
  "desired": {
    "COLOR": "RED",
    "MODE": "OFF"
  }
}
```

MODE can be ON or OFF and COLOR can be RED, GREEN or BLUE

5.4 AWS Demo for iMX Linux / RaspberryPi

This demo demonstrates connection to AWS IoT Console using pre-provisioned device credentials and publish/subscribe procedure using MQTT.

5.4.1 Prerequisites

- AWS account
- SD Card image with SE050 Middleware pre-installed. The application is built on the iMX platform.
- IMX6UL-EVK platform or Raspberry pi connected to the Internet

5.4.2 Preparing the credentials and Provisioning the secure element

Use ssscli tool from iMX/Rpi platform

- 1) Complete [Section 7.3 Steps needed before running ssscli tool](#)
- 2) To create certificates on imx and Raspberry Pi, call:

```
cd simw-top/pycli/Provisioning/
python3 GenerateAWSCredentials.py
python3 ResetAndUpdate_AWS.py
```

- 3) Certificates and Keys are generated at `simw-top/pycli/Provisioning/aws`

Build the OpenSSL engine [Optional]

Note: This step is optional in case you are using a prepared SD card image from NXP.

The OpenSSL engine uses the sss abstraction layer to access the crypto services of the secure element, the implementation remains dependent on the secure element attached. The following illustrates compiling the OpenSSL engine for SE050 connected over I2C.

```
cd simw-top
python scripts/create_cmake_projects.py
cd ../simw-top_build/imx_native_se050_tloi2c
cmake --build .
make install
ldconfig /usr/local/lib
```

Note: Replace `imx_native_se050_tloi2c` with `raspbian_native_se050_tloi2c` when building for Raspberry Pi.

5.4.3 Run the example

- 1) Clone the Code

```
cd /simw-top/demos/linux/aws/
git clone https://github.com/aws/aws-iot-device-sdk-cpp.git
```

Note: If curl is not installed - run `sudo apt-get install libcurl4-openssl-dev`

- 1) Modify the `CMakeLists.txt` file under `samples/PubSub` so it ensures `OPENSSL_LOAD_CONF` is defined (see excerpt below):

```
if (UNIX AND NOT APPLE)
    ADD_DEFINITIONS (-DOPENSSL_LOAD_CONF)
    # Prefer pthread if found
    set (THREADS_PREFER_PTHREAD_FLAG ON)
    set (CUSTOM_COMPILER_FLAGS "-fno-exceptions -Wall -Werror")
elseif (APPLE)
    set (CUSTOM_COMPILER_FLAGS "-fno-exceptions -Wall -Werror")
elseif (WIN32)
    set (CUSTOM_COMPILER_FLAGS "/W4")
endif ()
```

- 1) Use ‘buildScript.sh’ script at `simw-top/demos/linux/aws/` to build the mqtt application for aws call:

```
./buildScript.sh
```

- 1) Adapt the PubSub example specific configuration file so that it refers to the reference key and the device certificate.

- Update the endpoint to match your AWS account
- Ensure the `AmazonRootCA1.pem` certificate is in place (it is used by the iMX/rpi to validate the AWS IoT counterpart)
- Update the configuration file (`/simw-top/demos/linux/aws/aws-iot-device-sdk-cpp/build/bin/config/SampleConfig.json`) with `endpoint`, `device_certificate_relative_path`, `device_private_key_relative_path` (Ensure the value for “endpoint” matches your setup, you must replace “xxxxiukfoyyy-ats.iot.eu-central-1.amazonaws.com”)
- Sample Json file

```
{
  "endpoint": "xxxxiukfoyyyy-ats.iot.eu-central-1.amazonaws.com",
  "mqtt_port": 8883,
  "https_port": 443,
  "greengrass_discovery_port": 8443,
  "root_ca_relative_path": "certs/AmazonRootCA1.pem",
  "device_certificate_relative_path": "<UID>_device_certificate.crt",
  "device_private_key_relative_path": "<UID>_device_reference_key.pem",
  "tls_handshake_timeout_msecs": 60000,
  "tls_read_timeout_msecs": 2000,
  "tls_write_timeout_msecs": 2000,
  "aws_region": "",
  "aws_access_key_id": "",
  "aws_secret_access_key": "",
  "aws_session_token": "",
  "client_id": "CppSDKTesting",
  "thing_name": "CppSDKTesting",
  "is_clean_session": true,
  "mqtt_command_timeout_msecs": 20000,
  "keepalive_interval_secs": 600,
  "minimum_reconnect_interval_secs": 1,
  "maximum_reconnect_interval_secs": 128,
  "maximum_acks_to_wait_for": 32,
  "action_processing_rate_hz": 5,
  "maximum_outgoing_action_queue_length": 32,
  "discover_action_timeout_msecs": 300000
}
```

- 1) Search for *default_algorithms* in /simw-top/demos/linux/common/openssl_sss_se050.cnf file and set it as

```
default_algorithms = RSA,RAND,ECDSA,ECDH ----- For openssl 1.0.0
default_algorithms = RSA,RAND,EC ----- For openssl 1.1.1
```

- 1) Set the openssl config path as call:

```
$ export OPENSSL_CONF=/simw-top/demos/linux/common/openssl_sss_se050.cnf
```

- 2) Upload the root certificate (/simw-top/pycli/Provisioning/aws/rootCA_certificate.cer) to AWS account. Refer [Creating a device on AWS account](#)
- 3) Run the application:

```
cd /simw-top/demos/linux/aws/aws-iot-device-sdk-cpp/build/bin
./pub-sub-sample
```

Note:

- 1) Export the OpenSSL conf path to the exact location of the file. The above example is for illustrative purpose
-

5.5 GCP Demo for KSDK

This demo demonstrates connection to Google Cloud Platform using pre-provisioned device credentials and publish/subscribe procedure using MQTT.

5.5.1 Prerequisites

- Active azure account
- MCUXpresso installed (for running azure demo on k64)
- Any Serial communicator
- Flash VCOM binary on the device. VCOM binary can found under <PROJECT>binaries folder.
- Refer to *CLI Tool* for pyCLI tool setup

5.5.2 Using WiFi with LPC55S

WiFi shield Silex-2401 is supported with LPC55S. Mount the WiFi shield on to the arduino stackable headers.

5.5.3 Creating and updating device keys and certificates to SE

- 1) Complete *Section 7.3 Steps needed before running ssscli tool*
- 2) Check the vcom port number
- 3) To create certificates on windows and provision, go to `simw-top/pycli` directory and call:

```
call venv\Scripts\activate.bat
cd Provisioning
python GenerateGCPCredentials.py <COM_PORT>
python ResetAndUpdate_GCP.py <COM_PORT>
```

- 4) Certificates and Keys are generated at `simw-top/pycli/Provisioning/gcp`

5.5.4 Preparing the Cloud

- 1) Updating cloud over command line. Set up `gcloud` command line utility available at <https://cloud.google.com/pubsub/docs/quickstart-cli>
- Assuming the project name is `pgh-cloud-iot` the following commands sets up the code.

Create a new events pub/sub topic:

```
-----
| gcloud pubsub topics create a7lch-demo-events \
|   --project=pgh-cloud-iot
|
|-----
```

- Create a registry:

```

,-----
|
| gcloud iot registries create nxp-se-demo-reg \
|   --project=pgh-cloud-iot \
|   --region=us-central1 \
|   --event-notification-config=topic=projects/pgh-cloud-iot/topics/a71ch-demo-
|   ↪events \
|
|-----

```

- Create a device and attach the certificate `tls_client.cer`:

```

,-----
|
| gcloud iot devices create nxp-ecc-dev-01 \
|   --project=pgh-cloud-iot \
|   --region=us-central1 \
|   --registry=nxp-se-demo-reg \
|   --public-key=path=/simw-top/pycli/Provisioning/gcp/<UID>_device_certificate.
|   ↪cer,type=es256-pem
|
|-----

```

2) Updating cloud using the Web Interface

- A) Sign up for Google Cloud Platform - IoT (If you have not done that already)
- B) Create Registry & Device in the cloud platform.
- C) Copy For the device, add public key in ES256_X509 format Copy `hostLibmbedtlssecctls_client.cer` and paste in the web-dialogue box.

5.5.5 Running the Demo

- 1) Open `frdmk64f_mbedtls_aws_demo` project found under `<PROJECT>projects` in MCUXPRESSO IDE
- 2) Build the project and flash the binary on FRDM-K64F board
- 3) Connect your board to open network
- 4) **Open a serial terminal on PC for OpenSDA serial device with these settings:**
 - 115200 baud rate
 - 8 data bits
 - No parity
 - One stop bit
 - No flow control
 - change Setup->Terminal->New-line->Receive->AUTO
- 5) Console output - If everything is setup correctly the output would be as follows

```

,-----
|
| GCP JWT NXP Secure Element example

```

(continues on next page)

(continued from previous page)

```
|
| selectResponseDataLen: 2
| 0x01:0x31:
| Associating ECC key-pair '0'.
| Connecting to network
| Getting IP address from DHCP ...
|
| IPv4 Address      : 192.168.1.55
| DHCP OK
| Current EPOCH = 1520599186
| Using ECC key '0' for signing.
| JWT TOKEN = eyJhbGciOiJFUzI1NiIsInR5cCI6IkpXVCJ9.
| eyJhdWQiOiJwZ2gtY2xvdWQtaW90IiwiaWF0IjoxNTIwNTk5MTg2LCJleHAiOjE1MjA2MzUxODZ9.
| pZK9NjzD2rMdsU9H6bLPHNTsjHE77zHTMNHxVDVR3fYo39ttM2gYrhvJBR2Ct-9a2o8FwFqWjR8YY_
| lDwGjYyg
| GAE subscribe publish example
|
| Connecting...
| Associating ECC key-pair '0'.
| Using ECC key '0' to compute shared secret.
| Subscribing...
| -->sleep
| -->sleep
| Publish done
|
| Subscribe callback
|
| ...
| ...
|
|`-----`
```

- 6) You can update device config with following messages to toggle on-board keys. Using the below command, we can toggle LEDs:

```
|`-----`
|
| gcloud iot devices configs update \
|   --project=pgh-cloud-iot \
|   --region=us-central1 \
|   --registry=nxp-se-demo-reg \
|   --device=nxp-ecc-dev-01 \
|   --config-data='{ "red": "off" }'
|
|`-----`
```

User can toggle individual LEDs:

```
|`-----`
| { "green": "toggle", "user": "test1" }
| { "green": "on", "user": "test1" }
| { "red": "off", "user": "test1" }
|
|`-----`
```

For DOS Batch files, the commands can be like below (with escaping):

```

| -----
| gcloud iot devices configs update ^
|   --project=pgh-cloud-iot ^
|   --region=us-centrall ^
|   --registry=nxp-se-demo-reg ^
|   --device=nxp-ecc-dev-01 ^
|   --config-data="{\"red\":\"on\", \"blue\":\"off\", \"green\":\"off\"}^"
|
| gcloud iot devices configs update ^
|   --project=pgh-cloud-iot ^
|   --region=us-centrall ^
|   --registry=nxp-se-demo-reg ^
|   --device=nxp-ecc-dev-01 ^
|   --config-data="{\"red\":\"off\", \"blue\":\"on\", \"green\":\"off\"}^"
|
| gcloud iot devices configs update ^
|   --project=pgh-cloud-iot ^
|   --region=us-centrall ^
|   --registry=nxp-se-demo-reg ^
|   --device=nxp-ecc-dev-01 ^
|   --config-data="{\"red\":\"off\", \"blue\":\"off\", \"green\":\"on\"}^"
| -----

```

5.5.6 Appendix

1. For more information, refer to <https://github.com/GoogleCloudPlatform/cpp-docs-samples/tree/master/iot/mqtt-ciote>

5.6 GCP Demo for iMX Linux / Raspberry Pi

This demo demonstrates connection to Google Cloud Platform using pre-provisioned device credentials and publish/subscribe procedure using MQTT.

5.6.1 Prerequisites

- GCP account
- SD Card image with SE050 Middleware pre-installed. The application is built on the iMX platform.
- IMX6UL-EVK/RPi platform connected to the Internet
- Install autoconf and libtool. Execute - `sudo apt-get install autoconf libtool`

For additional information:

- Refer to *Development Platforms* for hardware setup and iMX setup
- Refer to *CLI Tool* for pyCLI tool setup

5.6.2 Preparing the credentials and Provision the SE

- 1) Complete [Section 7.3 Steps needed before running ssscli tool](#)
- 2) To create certificates on imx and Raspberry Pi, call:

```
cd simw-top/pycli/Provisioning
python3 GenerateGCPCredentials.py
python3 ResetAndUpdate_GCP.py
```

- 3) Certificates and Keys are generated at `simw-top/pycli/Provisioning/gcp`

Build the OpenSSL engine [Optional]

Note: This step is optional in case you are using a prepared SD card image from NXP.

The OpenSSL engine uses the sss abstraction layer to access the crypto services of the secure element, the implementation remains dependent on the secure element attached. The following illustrates compiling the OpenSSL engine for SE050 connected over I2C.

```
cd simw-top
python scripts/create_cmake_projects.py
cd ../simw-top_build/imx_native_se050_tloi2c
cmake --build .
make install
ldconfig /usr/local/lib
```

Note: Replace `imx_native_se050_tloi2c` with `raspbrian_native_se050_tloi2c` when building for Raspberry Pi.

5.6.3 Building the application

- 1) Use 'buildScript.sh' script at `simw-top/demos/linux/gcp/` to download all dependencies and build the mqtt application for gcp call:

```
cd /simw-top/demos/linux/gcp/
./buildScript.sh
```

- 2) Search for `default_algorithms` in `/simw-top/demos/linux/common/openssl_sss_se050.cnf` file and set it as

```
default_algorithms = RSA,RAND,ECDSA,ECDH      ----- For openssl 1.0.0
default_algorithms = RSA,RAND,EC               ----- For openssl 1.1.1
```

- 3) Set the openssl config path. Skip if already done:

```
$ export OPENSSL_CONF=/simw-top/demos/linux/common/openssl_sss_se050.cnf
```

- 4) Upload the root certificate (`/simw-top/pycli/Provisioning/gcp/rootCA_certificate.cer`) and device certificate (`/simw-top/pycli/Provisioning/gcp/<UID>_device_certificate.cer`) to GCP account. Refer [Preparing the Cloud](#). Skip if already done.

5) Run the application call:

```
cd /simw-top/demos/linux/gcp/gcp/cpp-docs-samples/iot/mqtt-ciote
$ ./mqtt_ciote --deviceid "nxp-ecc-dev-01" --region "us-central1" --registryid_
↪ "nxp-se-demo-reg" --projectid "pgh-cloud-iot" --keypath /simw-top/pycli/
↪ Provisioning/gcp/<UID>_device_reference_key.pem --rootpath /simw-top/demos/
↪ linux/gcp/keys/roots.pem --algorithm ES256
```

Note:

1. The above example is for illustrative purpose
 2. Export the open ssl conf path to the exact location of the file.
 3. While executing the application, use the appropriate values for registryid, projectid, keypath, rootpath and algorithm
-

5.6.4 Appendix

1. For more information, refer to <https://github.com/GoogleCloudPlatform/cpp-docs-samples/tree/master/iot/mqtt-ciote>

5.7 IBM Watson Demo for KSDK

This demo demonstrates connection to IBM Watson IoT platform using pre-provisioned device credentials and publish/subscribe procedure using MQTT

5.7.1 Prerequisites

- Active IBM Watson account
- MCUXpresso installed (for running azure demo on k64)
- Any Serial communicator
- Flash VCOM binary on the device. VCOM binary can found under <PROJECT>binaries folder.
- Refer to *CLI Tool* for pyCLI tool setup

5.7.2 Using WiFi with LPC55S

WiFi shield Silex-2401 is supported with LPC55S. Mount the WiFi shield on to the arduino stackable headers.

5.7.3 Setting up IBM Watson IoT Platform

1. Create an IBM ID which enables you to create an service instance for the Watson IoT platform (<https://idaas.iam.ibm.com/idaas/mtfim/sps/authsvc?PolicyId=urn:ibm:security:authentication:asf:basicldapuser>)
2. Create an instance of Internet of Things Platform in the Watson IoT after logging in.
3. Register the Root CA certificate (and Intermediate CA certificates if applicable) by following the below link https://console.bluemix.net/docs/services/IoT/reference/security/set_up_certificates.html#set_up_certificates
 - i) Click on 'Launch' button to access to the IoT dashboard
 - ii) Click on 'Settings' tab
 - iii) Click on 'CA Certificates'
 - iv) Click on 'Add Certificate'
 - v) A new pop-up appears. Click on 'Select a file' option and select the certificate
 - vi) Click on 'Save'
4. Configure the security policies of the service by following steps
 - i) Click on 'Security' tab
 - ii) Click on the pencil of 'Connection Security' to edit the preferences
 - iii) A new window is loaded, 'Connection Security'. In the Security Level field, select 'TLS with Client Certificate Authentication'.
 - iv) click on 'Save'
5. Register Device type
 - i) In the service, Click on 'Devices' tab and 'Device Types'
 - ii) Click on 'Add device type'.
 - iii) Select the 'Device' type and write a name. Device type shall be registered as 'NXP-SE050-EC-D' and Optionally, add a description.
 - iv) Click 'Next'
 - v) Optionally, add information to the rest of the fields. In this guide all the fields have been intentionally left empty. Finally, click 'Done'.
 - vi) For more information, refer to https://console.bluemix.net/docs/services/IoT/iotplatform_task.html#iotplatform_task
6. Register device
 - i) Click on 'Devices' tab.
 - ii) Click on 'Add Device'
 - iii) In the 'Identity' tab, select as 'Device Type' the one created in 'Register Device type' and the 'Device ID' the one retrieved using the pycli tool (UID of the device)
 - iv) Click on 'Next' button
 - v) Click Done button to create the device
7. Configure the application.
 - i) In the 'watson_iot_config.h' file, update the org details in the macro "WATSO-NIOT_MQTT_BROKER_ENDPOINT" which we get from the URL of the dashboard https://org_id.internetofthings.ibmcloud.com/dashboard/#/overview

- ii) update the org details and UID of the device in the macro ‘WatsonEchoCLIENT_ID’ which is available in ‘watson_iot_config.h’

8. Configuring the publish application

Create API key and authentication pair token

- i) In the Watson IoT Platform dashboard, go to Apps > API Keys.
- ii) Click Generate API Key.

Note: Important: Make a note of the API key and token pair. Authentication tokens are non-recoverable. If you lose or forget this token, you will need to re-register the API key to generate a new authentication token.

An example of an API key is a-organization_id-a84ps90Ajs

An example of a token is MP\$08VKz!8rXwnR-Q*

- iii) Add a comment to identify the API key in the dashboard, for example: Key to connect my application.
- iv) Click Finish

5.7.4 Creating and updating device keys and certificates to SE

- 1) Complete [Section 7.3 Steps needed before running ssscli tool](#)
- 2) Check the vcom port number
- 3) To create certificates on windows and provision, go to simw-top/pycli directory and call:

```
call venv\Scripts\activate.bat
cd Provisioning
python GenerateIBMCredentials.py <COM_PORT>
python ResetAndUpdate_IBM.py <COM_PORT>
```

- 4) Certificates and Keys are generated at simw-top/pycli/Provisioning/ibm

5.7.5 Running the Demo

1. Open <board>_mbedtls_sss_watson_demo project found under <PROJECT>/projects in MCUXPRESSO IDE
2. In the ‘watson_iot_config.h’ file, update the org details in the macro “WATSON_IOT_MQTT_BROKER_ENDPOINT” which we get from the URL of the dashboard https://org_id.internetofthings.ibmcloud.com/dashboard/#/overview
3. Update the org details and UID of the device in the macro ‘WatsonEchoCLIENT_ID’ which is available in ‘watson_iot_config.h’
4. In OrgDetails.cfg file, update the Org details which we got from the previous step in the ‘org’ section
5. In OrgDetails.cfg file, update the auth-key which we got from the above sections in to the ‘auth-key’ section
6. Update the UID of the device in the application test.py (at Line 40)
7. Build the project and flash the binary on FRDM-K64F board
8. Connect your board to open network

9. Open a serial terminal on PC for OpenSDA serial device with these settings:

- 115200 baud rate
- 8 data bits
- No parity
- One stop bit
- No flow control
- change Setup→Terminal→New-line→Receive→AUTO

10. Press reset button on the board

11. To see the event coming in to device and event going out of the device, login to the Watson IoT platform and launch the service: i) Click 'Devices' #) Click the registered device id #) Click 'Recent Events' #) Events will be displayed in portal

12. Persistent RED LED ON indicates error

13. All lights off along with the following message indicates readiness to subscribe messages from AWS:

```
Subscribing...
-->sleep
-->sleep
Publish done
```

14. Run the Publish application to publish events to the device.

To do that, run:

```
python test.py OrgDetails.cfg GREEN ON
```

The above command ensures that the green LED is turned ON. Similarly RED and BLUE LED can be turned ON and OFF

15. Events that are published shall be verified in the Watson Platform Dashboard(Refer to section 15)

5.7.6 Appendix

1. For more information, refer to https://cloud.ibm.com/docs/services/IoT?topic=iot-platform-about_iotplatform

5.8 IBM Watson Demo for iMX Linux / Raspberry Pi

This demo demonstrates connection to IBM Watson IoT platform using pre-provisioned device credentials and publish/subscribe procedure using MQTT

5.8.1 Prerequisites

- IBM Cloud account
- SD Card image with SE050 Middleware pre-installed. The application is built on the iMX platform.
- IMX6UL-EVK platform connected to the Internet

For additional information:

- Refer to *Development Platforms* for hardware setup and iMX setup
- Refer to *CLI Tool* for pyCLI tool setup

5.8.2 Preparing the credentials

- 1) Complete *Section 7.3 Steps needed before running ssscli tool*
- 2) To create certificates on imx and Raspberry Pi, call:

```
cd simw-top/pycli/Provisioning
python3 GenerateIBMCredentials.py
python3 ResetAndUpdate_IBM.py
```

- 3) Certificates and Keys are generated at `simw-top/pycli/Provisioning/ibm`

- The subject and subject alternative name of the device certificate must adhere to specific conventions. Both subject and subject alternative name contain the 10 byte UID value. In addition the Subject Alternative Name contains the device type.

5.8.3 Build the OpenSSL engine [Optional]

Note: This step is optional in case you are using a prepared SD card image from NXP.

The OpenSSL engine uses the sss abstraction layer to access the crypto services of the secure element, the implementation remains dependent on the secure element attached. The following illustrates compiling the OpenSSL engine for SE050 connected over I2C.

```
cd simw-top
python scripts/create_cmake_projects.py
cd ../simw-top_build/imx_native_se050_tloi2c
cmake --build .
make install
ldconfig /usr/local/lib
```

Note: Replace `imx_native_se050_tloi2c` with `raspbrian_native_se050_tloi2c` when building for Raspberry Pi.

5.8.4 Running the Demo on iMX/Raspberry Pi

- 1) Use 'buildScript.sh' script at <MW_SRC_DIR>/simw-top/demos/linux/ibm_watson_iot to download all dependencies and build the mqtt application for ibm watson call:

```
cd /simw-top/demos/linux/ibm_watson_iot
./buildScript.sh
```

- 2) Based on OpenSSL version and applicable Secure Element, select the appropriate configuration file in `<MW_SRC_DIR>/simw-top/demos/linux/common` directory:

```
openssl11l_sss_a71ch.cnf  ----- OpenSSL 1.1.1 and A71CH
openssl11l_sss_se050.cnf  ----- OpenSSL 1.1.1 and SE050
openssl_sss_a71ch.cnf     ----- OpenSSL 1.0.0 and A71CH
openssl_sss_se050.cnf     ----- OpenSSL 1.0.0 and SE050
```

- 3) Set the openssl config path. Skip if already done:

```
$ export OPENSSL_CONF=<MW_SRC_DIR>/simw-top/demos/linux/common/<appropriate-cnf-file>
```

- 4) Upload the root certificate (<MW_SRC_DIR>/simw-top/pycli/Provisioning/ibm/rootCA_certificate.crt) to your IBM account. Refer to [Setting up IBM Watson IoT Platform](#) for instructions on uploading the Root CA certificate and registering the device. Skip if already done.

- 5) Run the application in either of the following two ways:

- Parameters via commandline:

```
./watson_imx_linux --org <ORG> --keypath <MW_SRC_DIR>/simw-top/pycli/  
→ Provisioning/ibm/<UID>_device_reference_key.pem --devcert simw-top/pycli/  
→ Provisioning/ibm/<UID>_device_certificate.cer --topic_  
→ "iot-2/evt/status/fmt/json" --payload "{ \"d\" : : \  
→ { \"SensorID\" : : \"Test\", \"Reading\" : : 7 } } "
```

where *ORG* is the organization ID, *keypath* is the path to reference key corresponding to the device key and *devcert* is the path to device certificate.

- Parameters via json file:

```
./watson imx linux --json <input.txt>
```

Sample JSON file:

```
{  
  "hostname": "orgID.messaging.internetofthings.ibmcloud.com",  
  "protocol": "MQTTS",  
  "port": "8443",  
  "devcert": "  
→ "cert_0000000000000000000000000000000000000000000000000000000000000092.pem",  
  "keypath": "  
→ "keyref_0000000000000000000000000000000000000000000000000000000000000092.pem",  
→  
  "payload": "HelloMessage",  
  "topic": "iot-2/evt/status/fmt/string",  
  "rootpath": "rootCA.pem"  
}
```

Note:

- 1) The above example invocation is for illustrative purpose.
 - 2) Export the open ssl conf path to the exact location of the file.
 - 3) While executing the application, use the appropriate values for org, keypath and devcert.
-

5.8.5 Appendix

1. For more information, refer to https://cloud.ibm.com/docs/services/IoT?topic=iot-platform-about_iotplatform
2. <MW_SRC_DIR> is a placeholder for the path to the Plug & Trust MW. It would typically be /home/root/se050_mw_v02.08.00 (or a later version) on i.MX.

5.9 Azure Demo for KSDK

This demo demonstrates connection to Azure IoT Hub using pre-provisioned device credentials and demonstrates publish/subscribe procedure using MQTT.

5.9.1 Prerequisites

- Active azure account with iot hub created
- MCUXpresso installed (for running azure demo on k64)
- Any Serial communicator
- Flash VCOM binary on the device. VCOM binary can found under <PROJECT>/binaries folder.
- Refer to *CLI Tool* for pyCLI tool setup

5.9.2 Using WiFi with LPC55S

WiFi shield Silex-2401 is supported with LPC55S. Mount the WiFi shield on to the arduino stackable headers.

5.9.3 Creating a device on azure IoT Hub portal

1. Navigate to the Dashboard -> <Your IoT Hub> -> “IoT Devices”
2. Add a new device (e.g. device1), and for its authentication type choose X.509 CA Signed

Note: Creating a new device For large deployment of device, creating a device on azure iot hub can be done using Azure IoT service SDK

5.9.4 Creating and updating device keys and certificates to SE

- 1) Complete *Section 7.3 Steps needed before running ssscli tool*
- 2) Check the vcom port number
- 3) To create certificates on windows and provision, go to `simw-top/pycli` directory and call:

```
call venv\Scripts\activate.bat
cd Provisioning
python GenerateAZURECredentials.py <COM_PORT>
python ResetAndUpdate_AZURE.py <COM_PORT>
```

- 4) Certificates and Keys are generated at `simw-top/pycli/Provisioning/azure`

5.9.5 Uploading root certificates to IoT Hub

- 1) On Azure IoT Hub portal, Navigate to Dashboard --> <Your IoT Hub> --> Certificates. Click on Add
- 2) Enter a friendly name and upload the root certificate created in the previous step. Location - `simw-top/pycli/Provisioning/azure/RootCA.cer` -> Save
- 3) Your certificate will show in the Certificate Explorer list. Click on certificate added
- 4) In Certificate Details, click Generate Verification Code
- 5) The provisioning service creates a Verification Code that you can use to validate the certificate ownership. Copy the code to your clipboard
- 6) Use the `verification_certificate.py` to generate a verify certificate (`verifyCert4.cer`)

```
cd simw-top/pycli/Provisioning
python verification_certificate.py <RootCA_Certificate> <RootCA_Keypair>
↪<Verification Code>
```

- 7) On Azure portal -> Certificate Details, upload the `verifyCert4.cer` file generated and click Verify.
STATUS of your certificate should change to Verified in the Certificate Explorer list

5.9.6 Running the Demo

- 1) Open `frdmk64f_mbedtls_azure_demo` project found under `<PROJECT>projects` in MCUXPRESSO IDE
- 2) Update `AZURE_IOT_HUB_NAME` and `AZURE_DEVICE_NAME` in `<PROJECT>\demo\azure_demo\azure_iot_config.h` with your details
- 3) Build the project and flash the binary on FRDM-K64F board
- 4) Connect your board to open network
- 5) **Open a serial terminal on PC for OpenSDA serial device with these settings**
 - 115200 baud rate
 - 8 data bits
 - No parity

- One stop bit
- No flow control
- change Setup->Terminal->New-line->Receive->AUTO

6) Console Output

- Press reset button on the board If everything is setup correctly, the output would be as follows:

```

| -----
|
| Connecting to network
| Getting IP address from DHCP ...
|
| IPv4 Address      : 192.168.0.68
| DHCP OK
| MQTT attempting to connect to 'dev2'...
|
| Signing using key lX
| MQTT Echo demo subscribed to devices/dev2/messages/devicebound/#
| -->sleepEcho successfully published
| Echo successfully published
| -->sleepEcho successfully published
| Echo successfully published
|
| ...
| ...
|
| -----

```

- You can use device explorer tool to control the on-board LEDs. Open device explorer tool and update the IoT Hub connection string to connect to azure IoT hub, IoT Hub connection string is found at Azure IoT Hub -> Shared access policies -> iothubowner -> Connection String Primary key
- In the “Message To Devices” tab, select the device and send the message as:

```

| -----
| { "green": "toggle", "green": "on", "red": "off" }
| -----

```

5.9.7 Appendix

1. for more information, refer to <https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-mqtt-support>

5.10 Azure Demo for iMX Linux / Raspberry Pi

This demo demonstrates connection to Azure IoT Hub using pre-provisioned device credentials and demonstrates publish/subscribe procedure using MQTT.

5.10.1 Prerequisites

- Azure account
- SD Card image with SE050 Middleware pre-installed. The application is built on the iMX platform.
- IMX6UL-EVK platform or Raspberry pi connected to the Internet

5.10.2 Preparing the credentials and Provisioning the secure element

Use ssscli tool from iMX/Rpi platform

- 1) Complete [Section 7.3 Steps needed before running ssscli tool](#)
- 2) To create certificates on imx and Raspberry Pi, call:

```
cd simw-top/pycli/Provisioning
python3 GenerateAZURECredentials.py
python3 ResetAndUpdate_AZURE.py
```

- 3) Certificates and Keys are generated at `simw-top/pycli/Provisioning/azure`

5.10.3 Registering Device

To register the device onto the IoT Hub portal, we can either upload Root credentials manually or we can register an individual device using `azure_imx_register` application. If you wish to upload Root credentials, skip the next steps and proceed to [Uploading root certificates to IoT Hub](#).

5.10.4 Create device enrollment in azure IoT Hub portal

This step is only for individual device enrollment.

Prerequisite: Azure IOT hub and Azure IOT HUB DPS account which are linked.

Refer: <https://docs.microsoft.com/en-us/azure/iot-dps/tutorial-set-up-cloud>

<https://docs.microsoft.com/en-us/azure/iot-dps/quick-setup-auto-provision>

Once required accounts exist we can enroll the devices. For this we only need device certificate which we created in above steps.

Follow the steps to enroll the device: <https://docs.microsoft.com/en-us/azure/iot-dps/tutorial-provision-device-to-hub>

Note: When creating device certificates be sure to use only lower-case alphanumeric and hyphens in your device name.

Run `azure_imx_register` application to register the device onto your IoT Hub.

`azure_imx_register` application can take parameters either via JSON file or via command line. The required parameters are:

- `registerid`: Registration id of the device (common name of device certificate)
- `keypath`: Path to reference key pem file
- `devcert`: Path to device certificate
- `rootpath`: Path to azure root CA certificate

- idscope: IDScope (can found in Azure IoT-DPS account - Overview)

Run via command line as:

```
./azure_imx_register --registerid test-device --keypath keyref.pem --rootpath_
↪ azureRootCA.pem --devcert cert.pem --idscope 0ne00068F95
```

Or pass JSON file as:

```
./azure_imx_register --json json_register_config.json
```

Sample JSON file:

```
{
  "devcert": "cert.pem",
  "keypath": "keyref.pem",
  "id_scope": "0ne00068F95",
  "registration_id": "test-device",
  "rootpath": "azureRootCA.pem"
}
```

Upon successful registration, “DeviceID”.txt file is created with DeviceID, assigned hub along with keyref, device certificate and root certificate path. This file can be given as input to connect to device and send messages.

The device is now registered and appears on IoT Azure hub under devices tab

We can pass this JSON file to `azure_imx_connect` application to connect to IoT Hub. You can skip the next step and proceed to *Build the OpenSSL engine [Optional]*.

5.10.5 Uploading root certificates to IoT Hub

- 1) On Azure IoT Hub portal, Navigate to Dashboard --> <Your IoT Hub> --> Certificates. Click on Add
- 2) Enter a friendly name and upload the root certificate created in the previous step. Location - `simw-top/pycli/Provisioning/azure/RootCA.cer` -> Save
- 3) Your certificate will show in the Certificate Explorer list. Click on certificate added
- 4) In Certificate Details, click Generate Verification Code
- 5) The provisioning service creates a Verification Code that you can use to validate the certificate ownership. Copy the code to your clipboard
- 6) Use the `verification_certificate.py` to generate a verify certificate (`verifyCert4.cer`)

```
cd simw-top/pycli/Provisioning
python verification_certificate.py <RootCA_Certificate> <RootCA_Keypair>
↪ <Verification Code>
```

- 7) On Azure portal -> Certificate Details, upload the `verifyCert4.cer` file generated and click Verify.

STATUS of your certificate should change to Verified in the Certificate Explorer list

5.10.6 Build the OpenSSL engine [Optional]

Note: This step is optional in case you are using a prepared SD card image from NXP.

The OpenSSL engine uses the sss abstraction layer to access the crypto services of the secure element, the implementation remains dependent on the secure element attached. The following illustrates compiling the OpenSSL engine for SE050 connected over I2C.

```
cd simw-top
python scripts/create_cmake_projects.py
cd ../simw-top_build/imx_native_se050_tloi2c
cmake --build .
make install
ldconfig /usr/local/lib
```

Note: Replace `imx_native_se050_tloi2c` with `raspbrian_native_se050_tloi2c` when building for Raspberry Pi.

5.10.7 Run the example

- 1) Use ‘buildScript.sh’ script at `simw-top/demos/linux/azure/` to download all dependencies and build the mqtt application for azure call:

```
cd /simw-top/demos/linux/azure
./buildScript.sh
```

- 2) Based on OpenSSL version and applicable Secure Element, select the appropriate configuration file in `<MW_SRC_DIR>/simw-top/demos/linux/common` directory:

```
openssl11_sss_a71ch.cnf    ----- OpenSSL 1.1.1 and A71CH
openssl11_sss_se050.cnf   ----- OpenSSL 1.1.1 and SE050
openssl_sss_a71ch.cnf     ----- OpenSSL 1.0.0 and A71CH
openssl_sss_se050.cnf     ----- OpenSSL 1.0.0 and SE050
```

- 3) Set the openssl config path as call:

```
$ export OPENSSL_CONF=/simw-top/demos/linux/common/<appropriate-cnf-file>
```

- 4) To run the application, call:

```
$ ./azure_imx_connect --deviceid "<devive_name>" --keypath simw-top/pycli/
↪Provisioning/azure/<UID>_device_reference_key.pem --rootpath simw-top/demos/
↪linux/azure/azureRootCA.pem --devcert simw-top/pycli/Provisioning/azure/<UID>_
↪device_certificate.cer --hubname <IoTHubName>.azure-devices.net --username
↪<IoTHubName> --payload "<MESSAGE>"
```

Or pass JSON file as:

```
./azure_imx_connect --json json_connect_config.json
```

Sample `json_connect_config.json`:

```
{
  "assignedHub": "ABCD.azure-devices.net",
  "deviceId": "test-device",
  "registration_id": "test-device",
  "status": "assigned",
  "keypath": "keyref.pem",
  "devcert": "cert.pem",
  "rootpath": "azureRootCA.pem",
  "payload": "hello message from device test-device"
}
```

Note: If you have used `azure_imx_register` application, `json_connect_config.json` is same as `"DeviceID".txt`

Note:

- 1) Export the OpenSSL conf path to the exact location of the file. The above example is for illustrative purpose
 - 2) While executing the application, use the appropriate values for device cert, Device id, Path, hubname and user-name
-

5.11 Greengrass Demo for Linux

AWS IoT Greengrass is a software provided by AWS to extend cloud capabilities to locally connected devices. This allows local devices to publish/subscribe to a topic even if there is no connectivity with AWS IoT console. A Greengrass group consists of a Greengrass core, multiple Greengrass devices connected to that core, and lambda functions and other services running on that core. In this, the Greengrass core performs the functions of AWS IoT console.

Also see <https://docs.aws.amazon.com/greengrass/latest/developerguide/what-is-gg.html> for more details about AWS IoT Greengrass.

This demo is to demonstrate how to integrate SE050 with AWS IoT Greengrass core and RaspberryPi as hardware security to store core specific credentials for IoT client and MQTT server.

Note: Hardware security feature is available only for AWS IoT Greengrass Core v1.7 and later. We have used Greengrass core v1.9.2 for integration

5.11.1 Prerequisites

- AWS Greengrass account (Also see supported regions for Greengrass)
- RaspberryPi 3 Model B+ or Model B. The architecture of your Pi must be armv7l or later
- Raspbian Buster operating system
- Python 2.7
- pyCLI Tool. Refer to *CLI Tool*

5.11.2 Preparing the Greengrass group

- 1) Follow the modules 1 and 2 as described in <https://docs.aws.amazon.com/greengrass/latest/developerguide/module1.html> to set up Greengrass group and Greengrass core.

Note: In Module 2, if you choose Easy Group Creation, AWS will create credentials for Greengrass IoT core and provision in the registry. Skip the next step if you choose Easy Group Creation. You could otherwise create your own credentials and provision AWS registry as explained in the next step.

- 2) If you wish to use your own credentials, upload the your RootCA and verification certificate in Secure->CAs tab under IoT Core.
 - While creating Greengrass group, choose `Advanced group creation`.
 - You can either assign IAM role or skip it for later.
 - Under Set up your security, choose `Advanced setup` and then choose `Use my certificate`.
 - Select your active RootCA certificate and upload corresponding device certificate
- 3) If you used your own credentials, download sample `config.json` file for greengrass available at <https://docs.aws.amazon.com/greengrass/latest/developerguide/gg-core.html#config-json>

After completing Module 2, store your device certificate under `certs` directory where you have extracted AWS IoT Greengrass core software (by default `/greengrass` directory) and the downloaded `config.json` under `config` directory.
- 4) Do **NOT** run the daemon yet.

5.11.3 Provisioning SE050 and Building PKCS#11 library

- 1) Before running the Greengrass daemon, you would need to provision your SE050 and build PKCS#11 library.
- 2) Complete [Section 7.3 Steps needed before running ssscli tool](#) for pyCLI tool setup
- 3) Run the following steps to provision your SE050 with Greengrass core keypair:

```
ssscli connect se050 tloi2c none
ssscli se05x reset
ssscli set ecc pair 0x20181001 <path-to-core-keypair>
ssscli disconnect
```

Note: Greengrass uses labels to address objects on tokens. To make the PKCS#11 library use a specific keyID, the label should start with `sss:` followed by 32-bit keyID in hexadecimal format. For example, the label for the command used above would be `sss:20181001`.

- 4) Build PKCS#11 library for Greengrass core. Refer to [Section 6.7 PKCS#11 Standalone Library](#)

5.11.4 Updating Greengrass configuration

If you have successfully completed *Preparing the Greengrass group*, you would have `config.json` under `config` directory of AWS IoT Greengrass core software (by default as `/greengrass` directory). A sample of `config.json` is:

```
{
  "coreThing" : {
    "caPath" : "root.ca.pem",
    "certPath" : "hash.cert.pem",
    "keyPath" : "hash.private.key",
    "thingArn" : "arn:partition:iot:region:account-id:thing/core-thing-name",
    "iotHost" : "host-prefix-ats.iot.region.amazonaws.com",
    "ggHost" : "greengrass-ats.iot.region.amazonaws.com",
    "keepAlive" : 600
  },
  "runtime" : {
    "cgroup" : {
      "useSystemd" : "yes"
    }
  },
  "managedRespawn" : false,
  "crypto" : {
    "principals" : {
      "SecretsManager" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key"
      },
      "IoTCertificate" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key",
        "certificatePath" : "file:///greengrass/certs/hash.cert.pem"
      }
    },
    "caPath" : "file:///greengrass/certs/root.ca.pem"
  }
}
```

Remove the `caPath`, `certPath`, and `keyPath` values from the `coreThing` object.

Update the `certificatePath` property of `IoTCertificate` object to the path of device certificate.

Note: Currently AWS IoT Greengrass core does not support loading certificates from hardware. These have to be provided as a path to a file on filesystem.

Update the values of `privateKeyPath` under `SecretsManager` and `IoTCertificate` objects with `pkcs11:object=iotkey;type=private`.

Add the following `MQTTServerCertificate` object under `principals` object:

```
"MQTTServerCertificate": {
  "privateKeyPath": "pkcs11:object=iotkey;type=private"
}
```

Add the following `PKCS11` object under `crypto` object:

```
"PKCS11": {
  "P11Provider": "/path/to/libgreengrass.so",
  "slotLabel": "greengrass",

```

(continues on next page)

(continued from previous page)

```
"slotUserPin": "1234"
}
```

Add commas where needed to create a valid JSON document.

In this file, we have used a shared key for `MQTTServerCertificate`, `IoTCertificate` and `SecretsManager` components. In PKCS#11 object, we specify which PKCS#11 module to load and which slot to use in that module. All PKCS#11 objects specified for different components will refer to the same token.

5.11.5 Running Greengrass Core

Start the Greengrass daemon by running the following command in `ggc/core` directory under AWS IoT Greengrass core software directory:

```
sudo ./greengrassd start
```

The Daemon should start successfully. If you face any problem while starting the Greengrass daemon, refer to Troubleshooting section below. Also see runtime logs under `/greengrass/ggc/var/log/system` directory.

5.11.6 Connecting Devices to Greengrass Core

Follow steps mentioned from Module 3 to test Greengrass connectivity. <https://docs.aws.amazon.com/greengrass/latest/developerguide/module3-I.html>

5.11.7 Troubleshooting

1) Error message **greengrass deployment failed too many levels of symbolic links**

Check if your linux supports OverlayFS. Also confirm that the Raspberry Pi image version matches the version specified in <https://docs.aws.amazon.com/greengrass/latest/developerguide/setup-filter.rpi.html>. Currently, AWS IoT Greengrass Core has been tested on **2019-07-10-raspbian-buster** image. Greengrass core might not work with newer images like Raspbian Stretch.

2) Error message **connection reset by peer**.

Add properties `iotHttpPort` and `ggHttpPort` to `coreThing` object as:

```
"iotHttpPort" : 443,
"ggHttpPort"  : 443
```

If you face any other issue, refer to <https://docs.aws.amazon.com/greengrass/latest/developerguide/gg-troubleshooting.html>

5.12 OpenSSL Engine: TLS Client example for iMX/Rpi3

- DocRevision : 0.94
- Date : 2020-01-14

This section explains how to set-up a TLS link using the SE050 OpenSSL Engine on the client side. A note at the bottom of this page ([Section 5.12.7](#)) highlights the changes in case one uses an A71CH secure element.

5.12.1 Summary

The TLS demo demonstrates setting up a mutually authenticated and encrypted link between a client and a server system. The keypair used to identify the client is stored in the Secure Element. The keypair used to identify the server is simply available as a pem file.

The public keys associated with the respective key pairs are contained in respectively a client and a server certificate.

The CA is a self-signed certificate. The same CA is used to sign client and server certificate.

One can choose the keymaterial (CA, Client and Server) to be either RSA (4096 CA - 2048 client/server) or EC (prime256v1).

The TLS demo comes in two flavours:

1. **Flavour-A:** The standard OpenSSL tools *s_server* and *s_client* are used to set-up and demonstrate the TLS link. The certificates used are simply stored on the file system of the host. The client uses the keypair stored inside the SE050.
2. **Flavour-B:** A TLS client program - included in source code (*tlsSe050Client.cpp*) - retrieves the client certificate from the SE050 and uses the keypair stored inside the SE050. It establishes a TLS connection with the server process *s_server*.

Steps in [Section 5.12.2](#) to [Section 5.12.5](#) are identical for the two demo flavours.

5.12.2 Credential preparation (execute once) [Optional]

```
> cd demos/linux/tls_client
> ./scripts/createTlsCredentials_Optional.sh <ECC|RSA>
```

Note: The Host SW package comes bundled with the required credentials. The *createTlsCredentials_Optional.sh* script will re-create equivalent credentials (but with new/different keypairs)

The script creates all demo required client and server credentials on the client platform. One must transfer the server credentials to the server platform.

5.12.3 Secure Element preparation (client side)

For the purpose of the demo one MUST inject the TLS client key pair and certificate into the Secure Element and create a reference pem file referring to the provisioned key pair:

```
> cd demos/linux/tls_client/scripts
> python3 provisionTlsClient.py --key_type <ecc|rsa>
```

Further details on using these scripts can be found in the following:

provisionTlsClient.py

usage: `provisionTlsClient.py [-h] --key_type KEY_TYPE [--connection_data CONNECTION_DATA] [--connection_type CONNECTION_TYPE] [--subsystem SUBSYSTEM]`

Provision attached secure element with ECC/RSA keys

Preconditions:

- Secure element attached
- Virtual environment should be activated (not for iMX platform. Refer ssscli installation steps: Plug & Trust MW, Section 8.3 *Steps needed before running ssscli tool*)

Postconditions:

- Key pair injected on id referred by `KEYPAIR_INDEX_CLIENT_PRIVATE` variable
- Ref pem created
- Client certificate injected on id referred by `CERTIFICATE_INDEX` variable.

optional arguments:

-h, --help show this help message and exit

required arguments:

--key_type KEY_TYPE Supported key types => ecc, rsa

optional arguments:

--connection_data CONNECTION_DATA Parameter to connect to SE => eg. COM3, 127.0.0.1:8050, none. Default: none

--connection_type CONNECTION_TYPE Supported connection types => tloi2c, sci2c, vcom, jrcpv1, jrcpv2, pcsc. Default: tloi2c

--subsystem SUBSYSTEM Supported subsystem => se050, a71ch. Default: se050

Example invocation:

```
python provisionTlsClient.py --key_type ecc
python provisionTlsClient.py --key_type ecc --connection_data 169.254.0.1:8050
python provisionTlsClient.py --key_type rsa --connection_data 127.0.0.1:8050 --
→ connection_type jrcpv2
python provisionTlsClient.py --key_type rsa --connection_data COM3
python provisionTlsClient.py --key_type ecc --subsystem a71ch
```

5.12.4 Server side preparation

Note: The Host SW package comes bundled with the required server credentials.

Ensure the default server credentials or those created under (Section 5.12.2) are available on the server platform.

5.12.5 Start up the server

Note: The server can run e.g. on a PC. The server must be reachable over the TCP/IP network for the Client. Choose either a server using EC based credentials or a server using RSA based credentials.

Execute the following command on the server platform to use the EC based server credentials, make a note on the IP address of the server:

```
> cd demos/linux/tls_client/scripts
> ./tlsServer.sh <ECDHE|ECDHE_SHA256|max>
```

Execute the following command on the server platform to use the RSA based server credentials, make a note on the IP address of the server:

```
> cd demos/linux/tls_client/scripts
> ./tlsServer.sh RSA
```

5.12.6 Establish a TLS link from the client to the server

The client process establishing the TLS connection comes in two flavours: either *s_client* or a program provided in source code (tlsSe050Client.cpp). Invoke either example through a bash shell script.

Using *s_client*

Invoke the script using the IP address of the server as the first argument and ECDHE or ECDHE_SHA256 as the second argument (ECDHE corresponding to ECDH ephemeral) when connecting to a server using EC based credentials:

```
> ./tlsSeClient.sh <server-IP-address> <ECDHE|ECDHE_256>
```

Invoke the script using the IP address of the server as the first argument and RSA as the second argument when connecting to a server using RSA based credentials:

```
> ./tlsSeClient.sh <server-IP-address> RSA
```

In case OpenSSL 1.1.1 is available on *both* Client (i.MX or Raspberry Pi) and Server side, it's possible to request the usage of the TLS1.3 protocol (by default TLS1.2 is used). This is achieved by setting the environment variable REQ_TLS to tls1_3:

```
> REQ_TLS=tls1_3 ./tlsSeClient.sh <server-IP-address> ECDHE
```

Using tlsSe050Client.cpp

First compile the client program. Ensure all required SE050 specific libraries and header files have been installed on the linux system. By default this example links to static libraries:

```
> cd demos/linux/tls_client/build
> cmake ../.
> cmake --build .
```

Invoke the script using the IP address of the server as argument:

```
> ./tlsExtendedSeClient.sh <server-IP-address> <EC|RSA>
```

Note: The environment variable REQ_TLS is not applicable to this example. In case OpenSSL 1.1.1 is available on the client, the actual TLS protocol version used will be negotiated to the highest version supported by both client and server. Otherwise TLS1.2 will be used.

5.12.7 TLS client example using A71CH

Introduction

The TLS client example can also be used in combination with an A71CH secure element (in EC mode only). The following steps are different:

```
- secure element preparation
- client program invocation
```

Secure Element preparation (client side)

Specify an additional option `--subsystem a71ch`:

```
> python3 provisionTlsClient.py --key_type ecc --subsystem a71ch
```

Invocation of client program

Set the environment variable REQ_SE to a71ch when invoking the client program:

```
> REQ_SE=a71ch ./tlsSeClient.sh 192.168.1.190 ECDHE
> ... or ...
> REQ_SE=a71ch ./tlsExtendedSeClient.sh 192.168.1.190 EC
```

5.13 OPC UA (Open62541) Demo

5.13.1 Supported Platforms

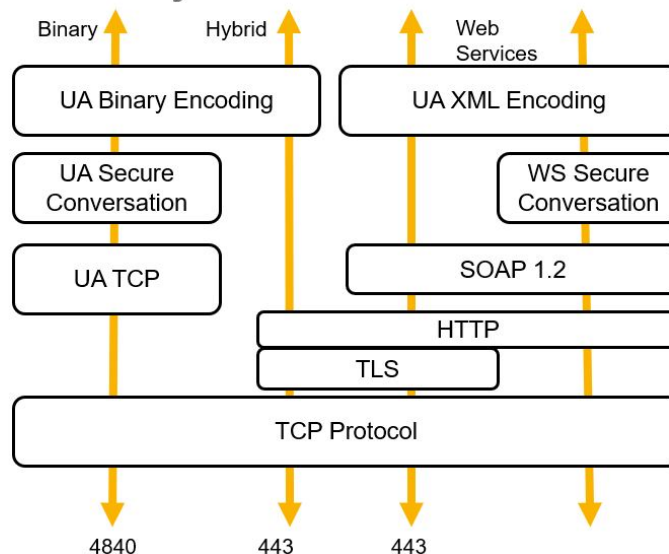
- Server Platform
 - Windows – JRCPv2 – SE050
 - iMX6 / RaspberryPi - t1oi2c – SE050
- Client Platform
 - UaExpert on Windows
 - Open62541 client on Windows
 - Open62541 client on iMX6 / RaspberryPi

5.13.2 Introduction

OPC UA (Open Platform Communications Unified Architecture) is an application layer protocol specific to Industrial IoT. It can run on top of TCP, TCP + Web services or TCP + HTTPS. In this client - server demo, the Open62541 open source OPC UA stack is used for integration with SE050. The server certificate and key are provisioned inside the SE050, the access to the SE050 is performed using the SSS APIs. The OPC UA server example source code is available in directory `demos\opc_ua\opc_ua_server`. The Open62541 specific adaptation layer to the SE050 is available in directory `sss\plugin\open62541`. The source code of the Open62541 stack is available in directory `ext\open62541`.

OPC UA stack:

OPC UA Security

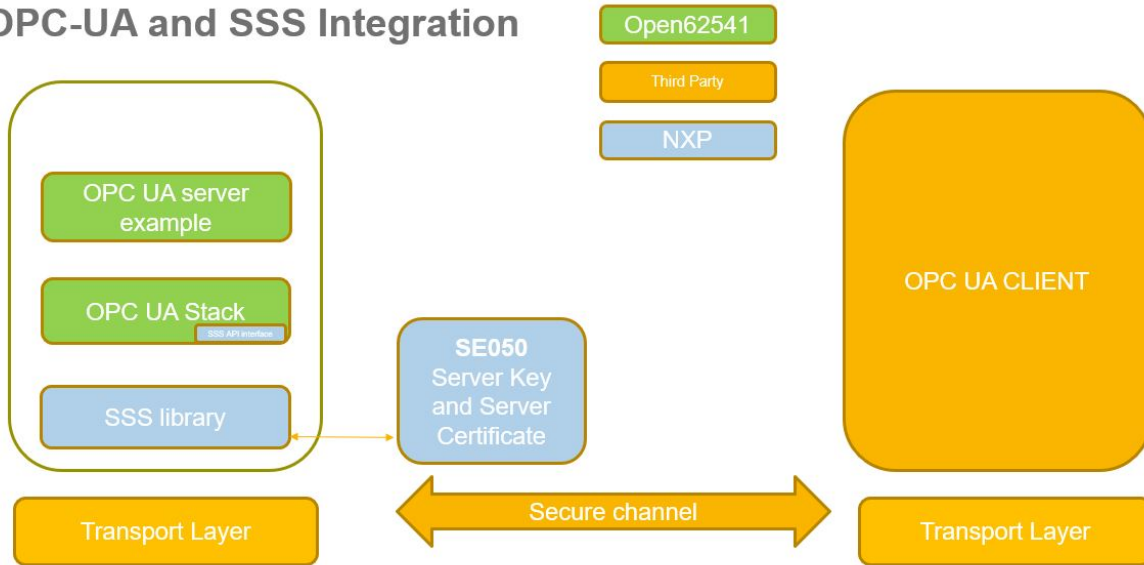


Note: more combinations are possible, e.g. JSON encoding is not shown

In reference to the above image the demo matches the left arrow:

- UA binary encoding is used
- UA Secure conversation with security policy `Basic256Sha256` and `Sign` and `Encrypt` mode
- on top of TCP

OPC-UA and SSS Integration



The crypto functionality (as defined by Basic256Sha256) is handled as follows:

- AsymmetricSignatureAlgorithm_RSA-PKCS15-SHA2-256: RSA Sign operation done by SE050
- AsymmetricEncryptionAlgorithm_RSA-OAEP-SHA: RSA Decrypt operation done by SE050
- Symmetric crypto operations are handled by the OPC UA stack on the host micro

5.13.3 Build Open62541 server and client examples

1) Build server and client example

```
cd simw-top
python3 scripts/create_cmake_projects.py
cd ../simw-top_build/imx_native_se050_t1oi2c
cmake -DWithOPCUA_open62541:BOOL=ON -DHostCrypto:STRING=MBEDTLS .
cmake --build .
make install
ldconfig /usr/local/lib
```

Note: Replace `imx_native_se050_t1oi2c` with `raspbrian_native_se050_t1oi2c` when building for Raspberry Pi.

1) Server and client binaries are copied to the simw-top/tools folder

5.13.4 Test Open62541 server and client examples

- 1) Client/Server keys are available in `simw-top\demos\opc_ua\credentials\`. Optionally you can regenerate the client/server keys with the following command

```
cd simw-top/demos/opc_ua/scripts
python3 createOPCUACredentials_Optional.py

OPU UA mandates the host name to be part of the subjectAltName in the server_
↳certificate.
The default server certificate provided with the package uses hostname 'localhost
↳'.
To create a completely new set of credentials with a specific server hostname /_
↳ip-address run
createOPCUACredentials_Optional.py script as

python3 createOPCUACredentials_Optional.py <server_hostname>    # Default <server_
↳hostname> = localhost
```

- 2) Refer to *CLI Tool* for pyCLI tool setup. Using pycli tool, provision server certificate and key into SE050 and create a reference pem file for server key

```
cd simw-top/demos/opc_ua/scripts

python3 provisionOPCUAServer.py 127.0.0.1:8050 jrcpv2    #On Windows
OR
python3 provisionOPCUAServer.py                        #On iMX6 / RaspberryPi
```

- 3) Start opc ua server

```
cd simw-top/demos/opc_ua/scripts

python3 open62541Server.py jrcpv2 127.0.0.1:8050 <certificate>    #On Windows
OR
python3 open62541Server.py <certificate>                        #On iMX6 /_
↳RaspberryPi

When using Open62541 client:
    <certificate> is located at simw-top\demos\opc_ua\credentials\open62541_
↳client_cert.der
When using UAexpert client:
    <certificate> is located at uaexpert\PKI\own\certs\uaexpert.der

Passing "none" for <certificate>, will make the server accept all client_
↳certificates.
```

- 4) Start opc ua client

```
cd simw-top/demos/opc_ua/scripts
python3 open62541Client.py opc.tcp://127.0.0.1:4840

On successful connection, value of the object "Sensor1" is read from server and_
↳displayed.
```

- 5) UaExpert client can also be used to test the Open62541 server.

- For testing with UaExpert client, root certificate needs to be copied to UaExpert trusted list of certificates,

- Go to UaExpert -> Settings -> Manage Certificates -> Trusted (Tab) -> Open Certificate Location and copy the file `simw-top\demos\opc_ua\credentials\open62541_rootCA_cert.der`
- **Also disable following errors in UaExpert configurations.**
 - i. UaExpert -> Settings -> Configure UaExpert -> General.DisableError.CertificateIssuerRevocationUnknown -> true
 - ii. UaExpert -> Settings -> Configure UaExpert -> General.DisableError.CertificateRevocationUnknown -> true
- **Add the server details to connect. UaExpert -> Server -> Add -> Advanced (Tab). Add details in**
 - i. EndPoint Url (`opc.tcp://<SERVER_IP>:4840/`)
 - ii. Security Policy as Basic256Sha256
 - iii. Message Security Mode as Sign & Encrypt
- Added server will appear in project tab. Right click on server -> Connect.
- On successful connection, the client objects should appear in UaExpert address space.
- To change the value of object “Sensor1”, select the object “Sensor1” in address space. In the Attribute section, select “value” attribute and enter the new value.

5.13.5 Known Limitations

- 1) Client certificates are self signed certificates. Not tested with root ca signed.
- 2) No root certificate can be given as input to command line Open62541 client. So any server certificate is accepted.

5.14 SE05X Minimal example

This project gets available memory from secure element. This is a good starting point to work with SE05X at low level.

5.14.1 Prerequisites

- Micro USB cable
- Kinetis FRDM-K64F/imx-RT1050 board
- Personal Computer
- SE05x Arduino shield

5.14.2 Building the Demo

- Build Plug & Trust middleware stack. (Refer *Building / Compiling*)
- Project: `se05x_minimal`

5.14.3 Running the Example

If you have built a binary, flash the binary on to the board and reset the board.

If you have built an *exe* to be run from Windows using VCOM, run as:

```
se05x_minimal.exe <PORT NAME>
```

Where **<PORT NAME>** is the VCOM COM port.

5.14.4 Console output

If everything is successful, the output will be similar to:

```
App      :INFO :mem=32767  
App      :INFO :ex_sss Finished
```

5.15 SE05X Get Info example

This project can be used to get SE05X platform information.

5.15.1 Prerequisites

- Micro USB cable
- Kinetis FRDM-K64F/imx-RT1050 board
- Personal Computer
- SE05x Arduino shield

5.15.2 Building the Demo

- Build Plug & Trust middleware stack. (Refer *Building / Compiling*)
- Project: se05x_Get_Info

5.15.3 Running the Example

If you have built a binary, flash the binary on to the board and reset the board.

If you have built an *exe* to be run from Windows using VCOM, run as:

```
se05x_Get_Info.exe <PORT NAME>
```

Where **<PORT NAME>** is the VCOM COM port.

5.15.4 Console output

If everything is successful, the output will be similar to:

```
App :WARN :#####
App :INFO :uid (Len=18)
      04 0D 0F 00      39 67 C2 C1      CE 41 1B 9A      76 F5 B7 AB
      D0 CD
App :WARN :#####
App :INFO :Applet Major = 3
App :INFO :Applet Minor = 1
App :INFO :Applet patch = 1
App :INFO :AppletConfig = 6FFF
App :INFO :With      ECDAA
App :INFO :With      ECDSA_ECDH_ECDHE
App :INFO :With      EDDSA
App :INFO :With      DH_MONT
App :INFO :With      HMAC
App :INFO :With      RSA_PLAIN
App :INFO :With      RSA_CRT
App :INFO :With      AES
App :INFO :With      DES
App :INFO :With      PBKDF
App :INFO :With      TLS
App :INFO :With      MIFARE
App :INFO :With      I2CM
App :INFO :Internal = 010B
App :WARN :#####
App :INFO :Tag value - proprietary data 0xFE = 0xFE
App :INFO :Length of following data 0x45 = 0x45
App :INFO :Tag card identification data (Len=2)
      DF 28
App :INFO :Length of card identification data = 0x42
App :INFO :Tag configuration ID (Must be 0x01) = 0x01
App :INFO :Configuration ID (Len=12)
      00 05 A2 00      F0 A4 91 BD      4B D8 24 3A
App :INFO :OEF ID (Len=2)
      A2 00
App :INFO :Tag patch ID (Must be 0x02) = 0x02
App :INFO :Patch ID (Len=8)
      00 00 00 00      00 00 00 00
App :INFO :Tag platform build ID1 (Must be 0x03) = 0x03
App :INFO :Platform build ID (Len=24)
      4A 33 52 33      35 31 30 32      31 45 45 45      30 34 30 30
      30 30 30 30      30 30 30 30
App :INFO :JCOP Platform ID = J3R351021EEE0400
App :INFO :Tag FIPS mode (Must be 0x05) = 0x05
App :INFO :FIPS mode var = 0x00
App :INFO :Tag pre-perso state (Must be 0x07) = 0x07
App :INFO :Bit mask of pre-perso state var = 0x00
App :INFO :Tag ROM ID (Must be 0x08) = 0x08
App :INFO :ROM ID (Len=8)
      52 4F 4D 5F      5F 5F 49 44
App :INFO :Status Word (SW) (Len=2)
      90 00
App :INFO :ex_sss Finished
```

5.16 APDU Player Demo

This demo is to transceive raw APDUs to the SE. It takes a command line argument as the input file containing APDUs to be sent, followed by the response. An example of command to be in the file is /send 00A4040000 6F108408A000000151000000A5049F6501FF9000

In this, the 00A4040000 is the command to be transmitted and 6F108408A000000151000000A5049F6501FF9000 is the expected response.

Note: Ensure that authentication with the SE is either plain or platform SCP This demo only supports only SE05x devices

5.16.1 How to use

Run the demo as apdu_player_demo [Input-file] <Port>

5.17 Using policies for secure objects

This demo is to demonstrate the use of policies for secure objects. Object policies such as can_Sign or can_Encrypt can be used to restrict operations other than the given policies. Objects inside the secure element are linked to a particular authentication object, based on communication authentication type selected. Objects inside the secure element linked with one authentication object cannot be used when session is open with another authentication type.

Authentication Object ID to be linked with secure object can be selected as

```
#if defined(EX_SE050_AUTH_UserID) || defined(EX_SE050_AUTH_UserID_PlatformSCP03) //
↪UserID Session
#define EX_LOCAL_OBJ_AUTH_ID EX_SSS_AUTH_SE05X_UserID_AUTH_ID
#elif defined(EX_SE050_AUTH_NONE) || defined(EX_SE050_AUTH_PlatformSCP03) //No auth
#define EX_LOCAL_OBJ_AUTH_ID EX_SSS_AUTH_SE05X_NONE_AUTH_ID
#elif defined(EX_SE050_AUTH_AESKey) || defined(EX_SE050_AUTH_AESKey_PlatformSCP03) //
↪AESKey
#define EX_LOCAL_OBJ_AUTH_ID EX_SSS_AUTH_SE05X_APPLETSCP_AUTH_ID
#elif defined(EX_SE050_AUTH_ECKey) || defined(EX_SE050_AUTH_ECKey_PlatformSCP03) //ECKey
↪session
#define EX_LOCAL_OBJ_AUTH_ID EX_SSS_AUTH_SE05X_ECKEY_ECDSA_AUTH_ID
#endif
```

Note: Ensure that the authentication object ID in policy set matches the authentication type.

5.17.1 Sign Policy

Create a policy set using the authentication object ID

```

/*Logic to pass sign policy*/
const int enable = 1;

/* doc:start:allow-policy-sign-part1 */
/* Policies for key */
const sss_policy_u key_withPol = {
    .type = KPolicy_Asym_Key,
    /*Authentication object based on SE05X_AUTH*/
    .auth_obj_id = EX_LOCAL_OBJ_AUTH_ID,
    .policy = {
        /*Asymmetric key policy*/
        .asymmkey = {
            /*Policy for sign*/
            .can_Sign = enable,
            /*Policy for verify*/
            .can_Verify = 1,
            /*Policy for encrypt*/
            .can_Encrypt = 1,
            /*Policy for decrypt*/
            .can_Decrypt = 1,
            /*Policy for Key Derivation*/
            .can_KD = 1,
            /*Policy for wrapped object*/
            .can_Wrap = 1,
            /*Policy to re-write object*/
            .can_Write = 1,
            /*Policy for reading object*/
            .can_Read = 1,
            /*Policy to use object for attestation*/
            .can_Attest = 1,
        }
    }
};

/* Common rules */
const sss_policy_u common = {
    .type = KPolicy_Common,
    /*Authentication object based on SE05X_AUTH*/
    .auth_obj_id = EX_LOCAL_OBJ_AUTH_ID,
    .policy = {
        .common = {
            /*Secure Messaging*/
            .req_Sm = 0,
            /*Policy to Delete object*/
            .can_Delete = 1,
            /*Forbid all operations on object*/
            .forbid_All = 0,
        }
    }
};

/* create policy set */
sss_policy_t policy_for_ec_key = {

```

(continues on next page)

(continued from previous page)

```

        .nPolicies = 2,
        .policies = { &key_withPol, &common }
    };
/* doc:end:allow-policy-sign-part1 */

/* doc:start:allow-policy-sign-part2 */
status = sss_key_store_generate_key(
    &pCtx->ks,
    &object,
    ECC_KEY_BIT_LEN,
    &policy_for_ec_key);
/* doc:end:allow-policy-sign-part2 */

```

5.17.2 Using PCR Object

PCR is a special secure object which stores 32-byte data. A PCR object can be used to ensure that secure objects inside the SE cannot be used if the PCR object value is altered.

We can assign a PCR policy to a secure object as given in the following sample code

```

uint8_t pcr_expected_value[] = { 0x89, 0x51, 0x56, 0x9f, 0x41, 0x5f, 0xeb, 0x4f, 0xb6,
    ↪ 0x37, 0x02, 0x86, 0xe7, 0xdd, 0xa0, 0x99, 0x33, 0x6c, 0x46, 0x36, 0xbc, 0xbb, 0x4c,
    ↪ 0x11, 0x04, 0x10, 0x0a, 0x86, 0x0d, 0x0c, 0xa4, 0x14 };
size_t pcr_expected_value_size = sizeof(pcr_expected_value);
LOG_I("Setting PCR Expected value as:");
LOG_AU8_I(pcr_expected_value, pcr_expected_value_size);

const sss_policy_u common = {
    .type = KPolicy_Common,
    .auth_obj_id = TST_LOCAL_OBJ_AUTH_ID,
    .policy = {
        .common = {
            .req_Sm = 0,
            .can_Delete = 1
        }
    }
};

const sss_policy_u file = {
    .type = KPolicy_File,
    .auth_obj_id = TST_LOCAL_OBJ_AUTH_ID,
    .policy = {
        .file = {
            .can_Read = 1,
            .can_Write = 1
        }
    }
};

sss_policy_u pcr1 = {
    .type = KPolicy_Common_PCR_Value,
    .auth_obj_id = TST_LOCAL_OBJ_AUTH_ID,
    .policy = {
        .common_pcr_value = {

```

(continues on next page)

(continued from previous page)

```
        .pcrObjId = 0x7fffffff,
    }
}
};
memset(pcr1.policy.common_pcr_value.pcrExpectedValue,
       0x00,
       sizeof(pcr1.policy.common_pcr_value.pcrExpectedValue));
memcpy(pcr1.policy.common_pcr_value.pcrExpectedValue, pcr_expected_value, pcr_
↪expected_value_size);

sss_policy_t policy_for_binary_object = {
    .nPolicies = 3,
    .policies = { &common, &pcr1, &file }
};
/* clang-format on */

status = sss_key_store_set_key(&gtCtx.ks,
    &gtCtx.key,
    binary_object,
    sizeof(binary_object),
    sizeof(binary_object),
    &policy_for_binary_object,
    sizeof(policy_for_binary_object));
```

Note: Ensure that the `pcrObjID` in PCR policy is the same object ID at which the PCR is stored.

5.17.3 Console output

If everything is successful, the output will be similar to:

```
App    :INFO :This example is to demonstrate the use of policies for secure objects
App    :INFO :Signing was succesful
App    :INFO :Example Success
App    :INFO :ex_sss Finished
```

5.18 Get Certificate from the SE

This tool is to retrieve Trust provisioned certificates from the SE. It will read the certificate and store it on the file system. It takes as argument, the keyID at which certificate is stored and the file path where to save the certificate on the file system.

Note: It can only be compiled when Host Crypto is mbedTLS.

5.18.1 Building the example

Use the following CMake configurations to compile the example

- CMake configurations: WithHostCrypto_MBEDTLS: ON
- Project: se05x_Get_Certificate

5.18.2 How to use

Run the tool as:

```
se05x_Get_Certificate <32-bit keyID> </path/to/file> <port>
```

- The keys ID has to be in HEX format.
- For systems connecting with T=1 over I2C, the port parameter can be skipped.
- The certificate will be stored in *PEM* format at the specified path on the file system.

5.19 SE05X Rotate PlatformSCP Keys Demo

This project is to demonstrate rotation of Platform SCP03 keys for IOT SSD. The Platform SCP03 keys used during initial authentication can be replaced using this example. In this example, we will rotate existing SCP03 keys to new keys and then revert back to the old keys.

Once the key rotation is successful on the IC a file is created `plain_scp.txt`. This file contains updated key values written to IC. For the next authentication if the above file is available, the keys are taken from the file. If the file is not present, the keys which are pre-compiled are picked up for authentication.

Following are the file paths for different platforms:

For Android

```
#define EX_SSS_SCP03_FILE_DIR "/data/vendor/SE05x/"
#define EX_SSS_SCP03_FILE_PATH EX_SSS_SCP03_FILE_DIR "plain_scp.txt"
```

For Linux

```
#define EX_SSS_SCP03_FILE_DIR "/tmp/SE05X/"
#define EX_SSS_SCP03_FILE_PATH EX_SSS_SCP03_FILE_DIR "plain_scp.txt"
```

For Windows

```
#define EX_SSS_SCP03_FILE_DIR "C:\\nxp\\SE05X\\"
#define EX_SSS_SCP03_FILE_PATH EX_SSS_SCP03_FILE_DIR "plain_scp.txt"
```

Note: For Android based platforms, it is important that the keymaster service has access to the PlatfSCP03 keys file while system boot. Be sure to update sepolicy accordingly.

5.19.1 Prerequisites

Since this example is portable across various platforms, the needs are different.

See Build Plug & Trust middleware stack. (Refer *Building / Compiling*)

5.19.2 Configuring the Demo

New Platform SCP03 keys are defined as following. Update your keys here.

```
#define EX_SSS_AUTH_NEW_ENC_KEY
↪      \
↪      {
↪      ↪      \
↪      ↪      0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, 0x4A, 0x4B, 0x4C, ↪
↪      ↪      0x4D, 0x4E, 0x4F \
↪      ↪      }

#define EX_SSS_AUTH_NEW_MAC_KEY
↪      \
↪      {
↪      ↪      \
↪      ↪      0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, 0x4A, 0x4B, 0x4C, ↪
↪      ↪      0x4D, 0x4E, 0x4F \
↪      ↪      }

#define EX_SSS_AUTH_NEW_DEK_KEY
↪      \
↪      {
↪      ↪      \
↪      ↪      0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, 0x4A, 0x4B, 0x4C, ↪
↪      ↪      0x4D, 0x4E, 0x4F \
↪      ↪      }
```

Old Platform SCP03 keys are defined as following. Make sure these match the ones in SE05X.

```
uint8_t OLD_KEY_ENC[] = EX_SSS_AUTH_SE05X_KEY_ENC;
uint8_t OLD_KEY_MAC[] = EX_SSS_AUTH_SE05X_KEY_MAC;
uint8_t OLD_KEY_DEK[] = EX_SSS_AUTH_SE05X_KEY_DEK;
```

The following code reverts to old Platform SCP03 keys. If you do not wish to revert to old keys and want to use the new keys, comment out the following line from the example. For development testing, we rollback to original keys. It is left to customer to comment out this line.

```
status = tp_PlatformKeys(OLD_KEY_ENC, OLD_KEY_MAC, OLD_KEY_DEK, pCtx);
```


5.19.3 Building the Demo

Use the following configurations in CMake:

- SE05X_Auth_PlatformSCP03: ON

Build project: se05x_RotatePlatformSCP03Keys

5.19.4 Running the Example

If you have built a binary, flash the se05x_RotatePlatformSCP03Keys binary on to the board and reset the board.

If you have built an *exe* to be run from PC using VCOM, run as:

```
se05x_RotatePlatformSCP03Keys.exe <PORT NAME>
```

Where <PORT NAME> is the VCOM COM port.

5.19.5 Console output

If everything is setup correctly the output would be as follows

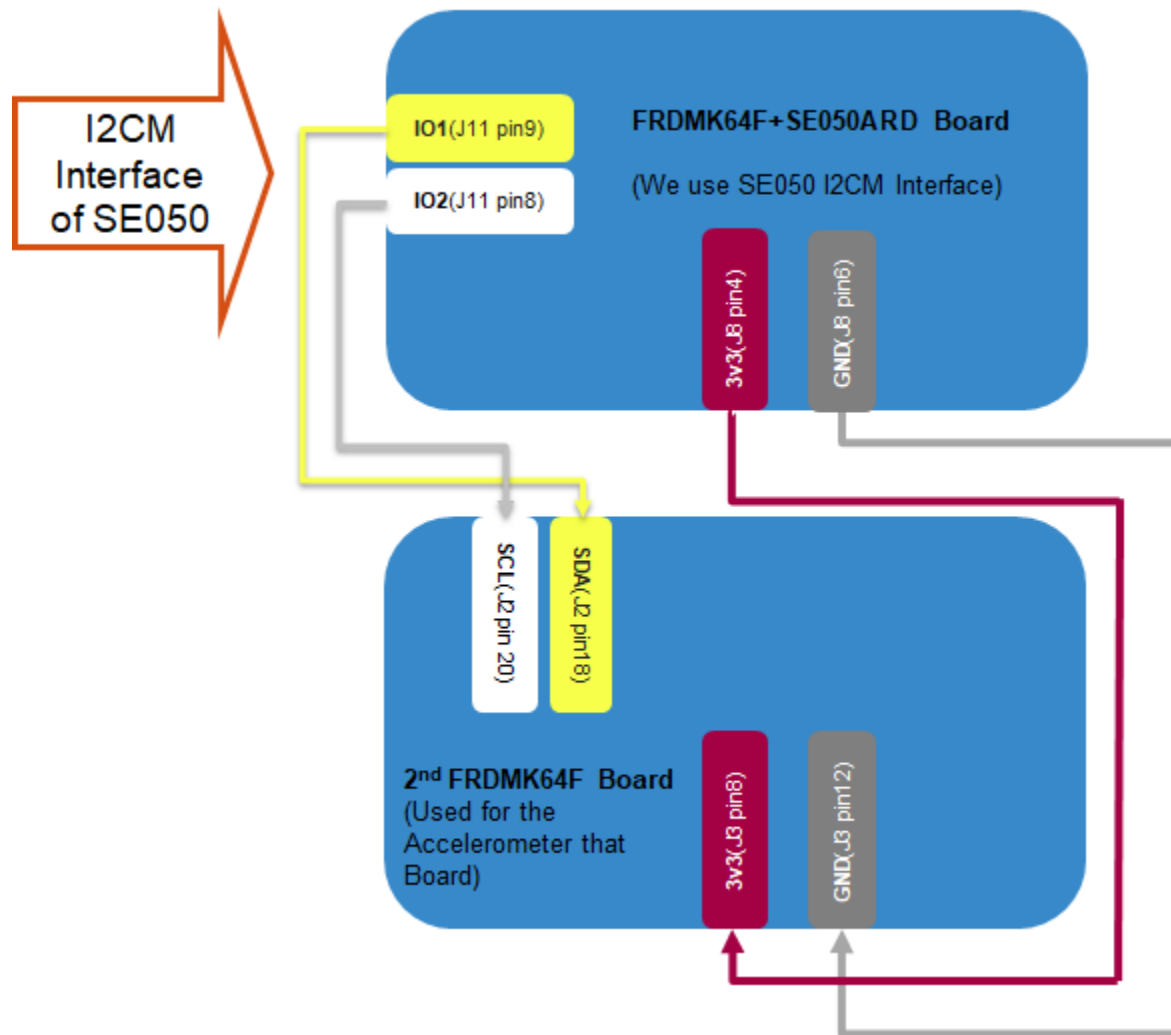
```
App:INFO :Congratulations !!! Key Rotation Successful!!!!  
App:INFO :Congratulations !!! Key Rotation Successful!!!!  
App:INFO :ex_sss Finished
```

5.20 I2C Master Example

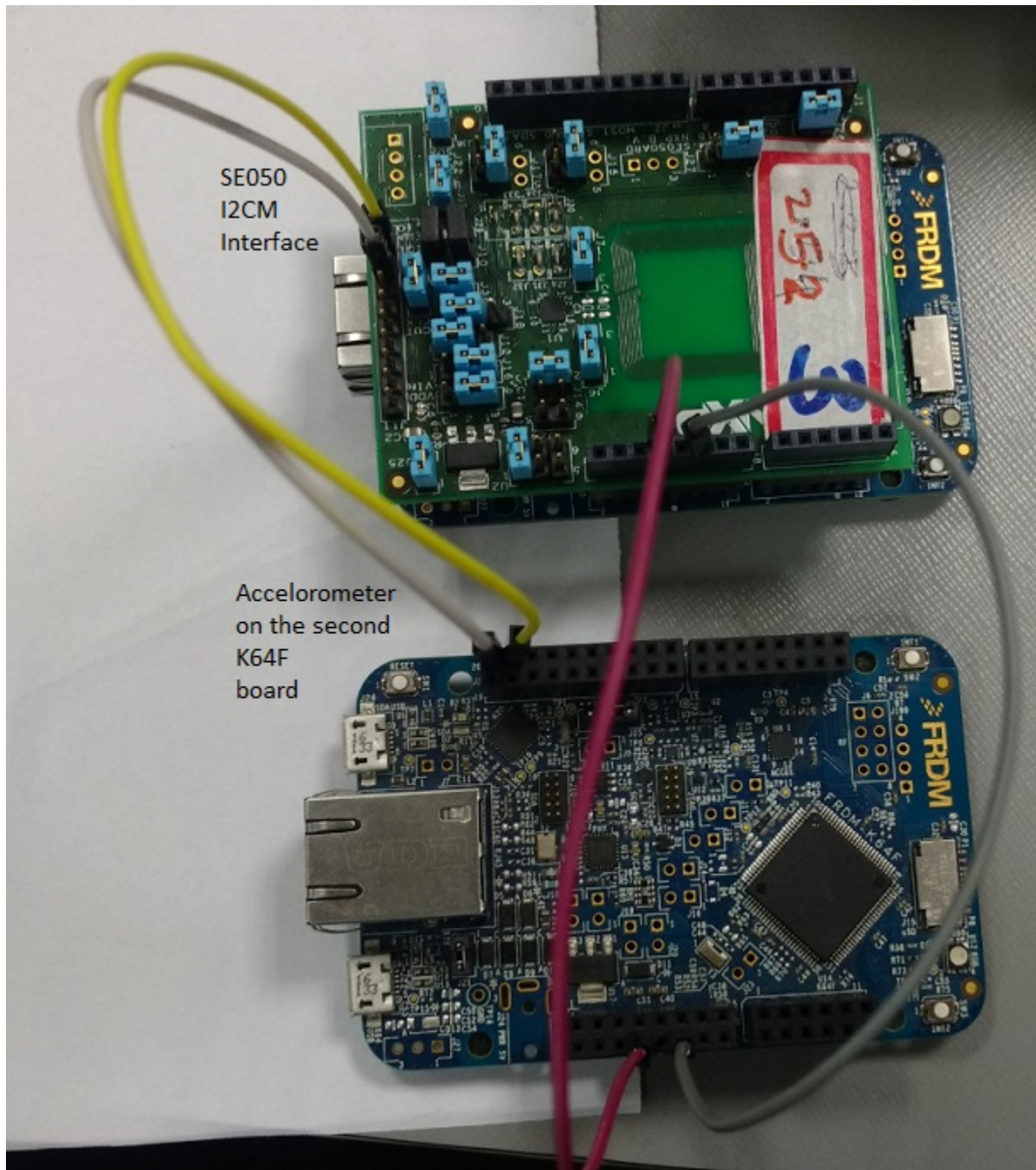
This page is regarding the documentation on I2CM, for more information on I2CM Transaction, See [Section 3.5 I2CM / Secure Sensor](#)

5.20.1 Prerequisites

- Bring Up Hardware. (Refer [Development Platforms](#))
- Connection to be done for I2C Master example.



Here is a photograph of above wiring diagram.



Note:

- We are using 2nd freedom K64F board only for connecting Accelerometer device to I2CM.
- Short Jumper J9 & J10 of se050ARD board.

Disable K64F on board 2

Warning: If the K64F of the 2nd board doing some operations on I2C Pins, this demo would not work.

Follow below steps to make this demo work.

- Flash `frdmk64f_nop_wfi.bin` binary located at `demos/se05x/ex_i2cMaster` directory to K64F board.
- This binary will put K64F in unoperation state and we will have easy access to Accelerometer through I2C pins.

Below is the c code

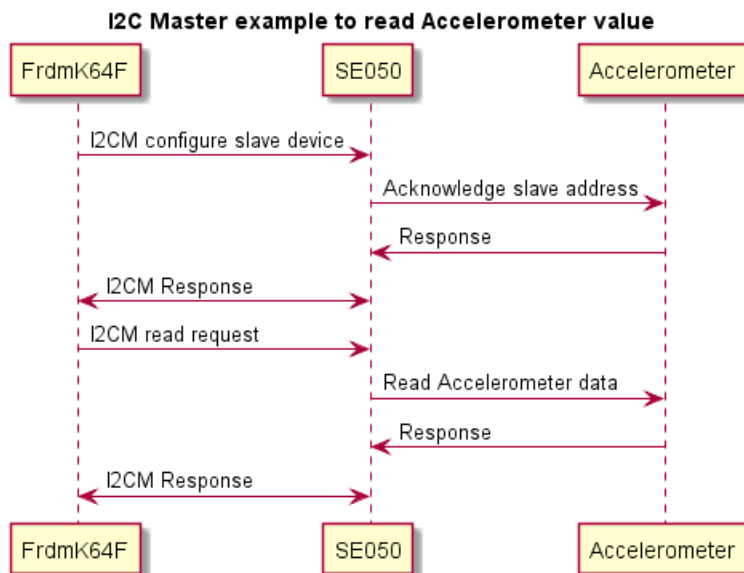
```
int main(void)
{
    /* Init board hardware. */
    BOARD_InitPins();
    BOARD_BootClockRUN();
    BOARD_InitDebugConsole();

    __disable_irq();
    while (1)
    {
        __disable_irq();
        __WFI();
    }
}
```

5.20.2 About the Example

This example reads Accelerometer data via the I2C master interface.

The Accelerometer on other K64F is used as an I2C Slave.



It uses the following APIs and data types:

- `Se05x_i2c_master_txn()`
- `Se05x_i2c_master_attst_txn()`
- `kSE05x_I2CM_Configure` from `SE05x_I2CM_TLV_type_t`
- `kSE05x_I2CM_Write` from `SE05x_I2CM_TLV_type_t`
- `kSE05x_I2CM_Read` from `SE05x_I2CM_TLV_type_t`
- `kSE05x_I2CM_Baud_Rate_400Khz` from `SE05x_I2CM_Baud_Rate_t`

5.20.3 Running the Demo

- 1) Import project `cmake_project_frdmk64f` from `simw-top/projects` directory.
- 2) Mention `BUILD_TARGET` as `ex_i2cMaster` or `ex_i2cMaster_with_Attestation` in `Debug/Makefile`.
- 3) Build the project and flash binary inside `FRDMK64F_SE050ARD` board.
- 4) Either press the reset button on your board or launch the debugger in your IDE to begin running the demo.
- 5) Rotate second K64F in any direction.

If everything is setup correctly the output would be as follows:

```
App:INFO :I2CM example to read Accelerometer value
App:INFO :x = 113 , y = -73 , z = 2118
App:INFO :x = 109 , y = -67 , z = 2103
App:INFO :x = 108 , y = -68 , z = 2120
App:INFO :x = 117 , y = -69 , z = 2109
App:INFO :x = 117 , y = -71 , z = 2105
App:INFO :x = 111 , y = -71 , z = 2108
App:INFO :x = 115 , y = -72 , z = 2104
App:INFO :x = 117 , y = -69 , z = 2122
App:INFO :x = 115 , y = -73 , z = 2120
App:INFO :x = 115 , y = -74 , z = 2114
App:INFO :I2CM test completed !!!...
```

5.21 SE05X WiFi KDF Example

This project is to demonstrate Password based KDF (PBKDF) operation using SE05X. This operation is used in deriving Pre-Shared key (PSK) for WiFi ssid using stored passwords.

5.21.1 Prerequisites

- Micro USB cable
- Kinetis FRDM-K64F/imx-RT1050 board
- Personal Computer
- SE05x Arduino shield

5.21.2 Building the Demo

- Build Plug & Trust middleware stack. (Refer *Building / Compiling*)
- Project: `ex_se05x_WiFiKDF_inject`
- Project: `ex_se05x_WiFiKDF_derive`

5.21.3 Running the Example

If you have built a binary, first flash the `ex_se05x_WiFiKDF_inject` binary on to the board and reset the board. Then flash `ex_se05x_WiFiKDF_derive` binary and reset the board.

If you have built an *exe* to be run from PC using VCOM, run as:

```
ex_se05x_WiFiKDF_inject.exe <PORT NAME>
ex_se05x_WiFiKDF_derive.exe -s <ssid_name> <PORT NAME>
```

Where **<PORT NAME>** is the VCOM COM port and **<ssid_name>** is the name of SSID for which you want to derive the PSK.

5.21.4 Console output

During injection, if everything is successful, the output will be similar to:

```
App :INFO :Injecting wifi_password='some-wifi-password'
App :INFO :ex_sss Finished
```

While deriving the key, if everything is successful, the output will be similar to:

```
App :INFO :Deriving PBKDF2 for wifi_ssid='some-wifi-ssid', WIFI_COUNT='4096'
App :INFO :wifi_derivedKey (Len=32)
      C9 A6 69 F9      6D A2 74 A1      41 43 A9 ED      D1 8F 68 1B
      B1 3E 6B 8B      F0 16 02 7A      7D 72 BF 0E      0C 53 CD 7C

# Data for /etc/wpa_supplicant/wpa_supplicant.conf
network={
    ssid="some-wifi-ssid"
    psk=c9a669f96da274a14143a9edd18f681bb13e6b8bf01627a7d72bfec53cd7c
}
App :INFO :Done
App :INFO :ex_sss Finished
```

5.22 SE05X Import Transient objects

Before running this example, please run [Section 5.23 SE05X Export Transient objects](#)

This example does following steps:

- Re-Generates a Transient ECC Key at the same location as created by *SE05X Export Transient objects*
- Tries to verify previously signed data. (This must fail, because key is over-written)
- Imports previously exported key inside the SE.
- Tries to verify previously signed data. (This must be successful, because it is the same key)

Note: This example needs File system access and hence it is not applicable for embedded platforms.

The exported files are current working directory.

5.22.1 Visual Studio - Debug settings

Kindly set debug Working Directory to \$(TargetDir) in project's debug settings. This ensures that the examples *SE05X Export Transient objects* and *SE05X Import Transient objects* work seamlessly.

5.22.2 Console output

If everything is successful, the output will be similar to:

```
App :INFO :Running Example ex_sss_import.c
App :INFO :Object exists!!!
App :INFO :This verify must fail, because keys are different
App :INFO :Reading contents form 'export_serializedSingedData.bin'
App :WARN :Verification Failed!
App :WARN :nxEnsure:'status == kStatus_SSS_Success' failed. At Line:196_
↪Function:ExampleDoVerify
App :INFO :Reading contents form 'export_serializedECKey.bin'
App :INFO :This verify must pass, because keys are same
App :INFO :Reading contents form 'export_serializedSingedData.bin'
App :INFO :Verification Successful.
App :INFO :ex_sss_import Example Success !!!...
App :INFO :ex_sss Finished
```

5.23 SE05X Export Transient objects

This example does following steps:

- Generate a Transient ECC Key
- Export that to a blob
- Sign some dummy data with that key
- Store the signature in local file for future use.

After running this example, you can run [Section 5.22 SE05X Import Transient objects](#)

Note: This example needs File system access and hence it is not applicable for embedded platforms.

The exported files are current working directory.

5.23.1 Visual Studio - Debug settings

Kindly set debug Working Directory to \$(TargetDir) in project's debug settings. This ensures that the examples *SE05X Export Transient objects* and *SE05X Import Transient objects* work seamlessly.

5.23.2 Console output

If everything is successful, the output will be similar to:

```
App :INFO :Running Example ex_sss_export.c
App :INFO :Export ec key to 'export_serializedECKey.bin'!!!
App :INFO :Signing Successful !!!
App :INFO :Export signature key to 'export_serializedSingedData.bin'.
App :INFO :ex_sss_export Example Success !!!...
App :INFO :ex_sss Finished
```

5.24 Import External Object Prepare

Import External Object command allows the user to import an external object wrapped with a secure ECKey_Auth context. A session is not required to execute this command, the ECKey_Auth parameters are provided with the wrapped WriteSecureObject command. The applet will use ECKey_Auth parameters and derive session keys to unwrap the command and execute it. ImportExternalObject command works in its own session. It will open an ECKey session, write the secure object and close the session.

In this example, we prepare a complete raw APDU to be sent to SE05x. A WriteSecureObject command needs to be prepared which will be wrapped and sent as a part of ImportExternalObject command. For an example we are preparing WriteSymmKey command as :

```
/* Symmetric Key */
/* clang-format off */
uint8_t keyValue[] = {0x48, 0x45, 0x4C, 0x4C, 0x4F, 0x48, 0x45, 0x4C, 0x4C, 0x4F,
↳ 0x48, 0x45, 0x4C, 0x4C, 0x4F, 0x31};
/* clang-format on */
/* API to create buffer */
pse05xWriteBufferSessionCtx->fp_TXn = &fLogTransmitBufferCreateAPDU;
int index = 0;
Se05xPolicy_t policy;
uint8_t policyBuffer[MAX_POLICY_BUFFER_SIZE] = {0};
size_t policyBufferLen = sizeof(policyBuffer);

/* Create policy if required */
getWriteOnlyKeyPolicy(policyBuffer, &policyBufferLen);
policy.value = policyBuffer;
policy.value_len = policyBufferLen;

sm_status = Se05x_API_WriteSymmKey(pse05xWriteBufferSessionCtx,
    &policy,
    SE05x_MaxAttempts_NA,
    __LINE__,
    SE05x_KeyID_KEK_NONE,
    keyValue,
    sizeof(keyValue),
    kSE05x_INS_NA,
    kSE05x_SymmKeyType_AES);
```

(continues on next page)

(continued from previous page)

```

if (sm_status != SM_OK) {
    LOG_E("Failed to create buffer");
    status = kStatus_SSS_Fail;
    goto exit;
}
/* WriteSecureObject API will prepare complete APDU.
 * We need to skip initial CLA INS P1 P2 and use just the TLVs
 *
 * The length is determined by the first length byte. If it
 * is 0x00, the next two bytes are the length, otherwise that
 * byte is the length.
 *
 * Determine the length here and accordingly determine the TLV.
 */
if (gTxBuffer[4] == 0x00) {
    WriteSymmKeyAPDU_len = ((gTxBuffer[5] << 8) && 0xFF00) | ((gTxBuffer[6]) && 0xFF);
    index = 7;
}
else {
    WriteSymmKeyAPDU_len = gTxBuffer[4];
    index = 5;
}
if (WriteSymmKeyAPDU_len > sizeof(WriteSymmKeyAPDU)) {
    LOG_E("Insufficient buffer");
    status = kStatus_SSS_Fail;
    goto exit;
}
memcpy(WriteSymmKeyAPDU, &gTxBuffer[index], WriteSymmKeyAPDU_len);

```

```

/* This API creates an APDU buffer and stores it to the global buffer */
smStatus_t fLogTransmitBufferCreateAPDU(Se05xSession_t *pwrite_apdubufferctx,
    const tlvHeader_t *hdr,
    uint8_t *cmdBuf,
    size_t cmdBufLen,
    uint8_t *rsp,
    size_t *rspLen,
    uint8_t hasle)
{
    memset(gTxBuffer, 0, sizeof(gTxBuffer));
    size_t i = 0;
    memcpy(&gTxBuffer[i], hdr, sizeof(*hdr));
    smStatus_t ret = SM_OK;
    i += sizeof(*hdr);
    if (cmdBufLen > 0) {
        // The Lc field must be extended in case the length does not fit
        // into a single byte (Note, while the standard would allow to
        // encode 0x100 as 0x00 in the Lc field, nobody who is sane in his mind
        // would actually do that).
        if ((cmdBufLen < 0xFF) && !hasle) {
            gTxBuffer[i++] = (uint8_t)cmdBufLen;
        }
        else {
            gTxBuffer[i++] = 0x00;
            gTxBuffer[i++] = 0xFFu & (cmdBufLen >> 8);
            gTxBuffer[i++] = 0xFFu & (cmdBufLen);
        }
    }
}

```

(continues on next page)

(continued from previous page)

```

        memcpy(&gTxBuffer[i], cmdBuf, cmdBufLen);
        i += cmdBufLen;
    }
    if (hasle) {
        gTxBuffer[i++] = 0x00;
        gTxBuffer[i++] = 0x00;
    }
    ret = SM_OK;
    gTxBufferLen = i;

    LOG_AU8_I(gTxBuffer, gTxBufferLen);

    return ret;
}

```

You can call any of the WriteSecureObject API with your data and create the buffer.

5.24.1 Building

Build the project with the following configurations.

se05x_ImportExternalObjectPrepare

- Project = se05x_ImportExternalObjectPrepare
- SCP=SCP03_SSS
- SE05x_Auth=ECKey or SE05x_Auth=ECKey_PlatformSCP03

5.24.2 How to use

Generate the raw APDU file by running the executable. Run **se05x_ImportExternalObjectPrepare** as

```
se05x_ImportExternalObjectPrepare.exe -keyid 0x7DA00003 -file eckey_ecdsa.der -out_
↪rawAPDU.der <portName>
```

where,

- *keyid* is the authentication keyId at which ECDSA public key is stored.
- *file* is the input ECDSA keypair file (in binary format)
- *out* is the output file where the raw APDU will be stored.
- *portName* is the name of the port over which to connect (COMPORT in case running over VCOM)

After this executes successfully, you need to send the generated raw APDU to SE05x. Refer to [Section 5.25 Import External Object Create](#)

5.25 Import External Object Create

In this example, we send a raw APDU to SE05x which will import the external object wrapped with ECKey context in the APDU.

Note: This APDU must be prepared by `se05x_ImportExternalObjectPrepare` executable.

5.25.1 Pre-requisites

Raw APDU should be prepared by `se05x_ImportExternalObjectPrepare`. Refer to [Section 5.24 Import External Object Prepare](#).

5.25.2 Building

Build the project with the following configurations.

se05x_ImportExternalObjectCreate

- Project = `se05x_ImportExternalObjectCreate`
- SCP=SCP03_SSS
- SE05x_Auth=None or SE05x_Auth=PlatfSCP03

5.25.3 How to use

Run executable **se05x_ImportExternalObjectCreate** as

```
se05x_ImportExternalObjectCreate.exe -file rawAPDU.der <portName>
```

where,

- *file* is the input file containing raw APDU to be sent. Same as the output file from `se05x_ImportExternalObjectPrepare`.
- *portName* is the name of the port over which to connect (COMPORT in case running over VCOM)

Note: `se05x_ImportExternalObjectCreate` example will fail if the object already exists.

5.26 SE05X Mandate SCP example

This project demonstrates how to configure SE05X to mandate platform SCP. This can be used if you always want the communication with SE05X to be encrypted.

Note: After this example runs successfully, further communication would require Platform SCP03 encryption.

5.26.1 Prerequisites

- Micro USB cable
- Kinetis FRDM-K64F/imx-RT1050 board
- Personal Computer
- SE05x Arduino shield

5.26.2 Building the Demo

- Build Plug & Trust middleware stack. (Refer *Building / Compiling*)
 - Project: se05x_MandatePlatformSCP
 - CMake configuration
- ```
SE05X_Auth:None
```

### 5.26.3 Running the Example

If you have built a binary, flash the binary on to the board and reset the board.

If you have built an *exe* to be run from Windows using VCOM, run as:

```
se05x_MandatePlatformSCP.exe <PORT NAME>
```

Where **<PORT NAME>** is the VCOM COM port.

## 5.27 Read object with Attestation

This example demonstrates how to read an object with attestation and parse the attested data to check various object attributes.

In this example, we use an EC NIST-P 256 keypair as the attestation key and a binary object which will be attested.

---

**Note:** The maximum size of a binary object that can be attested at a time is 500 bytes. The API available will only work on binary objects with size up to 500 bytes. To perform attestation on an object of greater size, we need to call corresponding Se05x API in a loop, verifying the obtained signature every time.

A reference implementation is available at *Reading large binary objects with attestation*

---

### 5.27.1 Building

Build the project with the following configurations.

**se05x\_ReadWithAttestation**

- Project = se05x\_ReadWithAttestation

## 5.27.2 Running

On running the example, you would be able to see object attributes logged on the screen like:

```
App :INFO :Running example se05x_ReadWithAttestation
App :INFO :Type:
App :INFO : BINARY_FILE
App :INFO :Auth:
App :INFO : Not Set
App :INFO :Neg auth count:
App :INFO : 0x0000
App :INFO :Owner:
App :INFO : 0x0000
App :INFO :Neg auth count max:
App :INFO : 0x0000
App :INFO :Auth Object:
App :INFO : No authentication required
App :INFO :Policies:
App :INFO : POLICY_OBJ_ALLOW_READ
App :INFO : POLICY_OBJ_ALLOW_WRITE
App :INFO : POLICY_OBJ_ALLOW_DELETE
App :INFO :Origin:
App :INFO : EXTERNAL
App :INFO :Example success
App :INFO :ex_sss Finished
```

You can see the various attributes associated with the object such as object type, authentication mechanism, origin and policies.

An example of how to perform read with attestation is given below

```
/* Prepare/init attestation data structure */

sss_se05x_attst_comp_data_t comp_data[2] = {0};
sss_se05x_attst_data_t att_data = {.valid_number = 2};
/* Random data from the host to check if SE
 * answers to current attestation request and
 * not an older response is used */

/* clang-format off */
uint8_t freshness[16] = { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b,
↪ 0x0c, 0x0d, 0x0e, 0x0f };
```

The data received in att\_data variable can be parsed to read the object attributes.

## 5.27.3 Reading large binary objects with attestation

Following is an example code on how to read a large binary file with attestation.

**Note:** This is required only when reading binary objects of size larger than 500 bytes. For any other case, you should use SSS API as above

```
sss_status_t read_large_object_with_attestation(ex_sss_boot_ctx_t *pCtx,
 sss_se05x_key_store_t *keyStore,
 sss_se05x_object_t *keyObject,
```

(continues on next page)

(continued from previous page)

```

uint8_t *key,
size_t *keylen,
size_t *pKeyBitLen,
sss_se05x_object_t *keyObject_attst,
sss_algorithm_t algorithm_attst,
uint8_t *random_attst,
size_t randomLen_attst,
sss_se05x_attst_data_t *attst_data)
{
 smStatus_t status = SM_NOT_OK;
 sss_status_t sss_status = kStatus_SSS_Fail;
 uint16_t rem_data = 0;
 uint16_t offset = 0;
 uint16_t size = 0;
 size_t max_buffer = 0;
 size_t signatureLen = 0;
 uint32_t attestID;
 SE05x_AttestationAlgo_t attestAlgo = kSE05x_AttestationAlgo_EC_SHA_256;
 attestID = keyObject_attst->keyId;
 /* Variables for verification */
 sss_object_t verification_object = {0};
 uint8_t plainData[2500] = {0};
 size_t plainDataLen = sizeof(plainData);
 uint8_t digest[64] = {0};
 size_t digestLen = sizeof(digest);
 sss_digest_t digest_ctx;
 sss_algorithm_t algorithm = kAlgorithm_SSS_ECDSA_SHA256;
 sss_algorithm_t digest_algorithm = kAlgorithm_SSS_SHA256;
 sss_asymmetric_t verify_ctx;

 if (kStatus_SSS_Success != create_host_public_key(pCtx, attestID, EC_KEY_BIT_LEN,
↪ &verification_object)) {
 goto cleanup;
 }

 status = Se05x_API_ReadSize(&keyStore->session->s_ctx, keyObject->keyId, &size);
 ENSURE_OR_GO_CLEANUP(status == SM_OK);

 if (*keylen < size) {
 LOG_E("Insufficient buffer ");
 goto cleanup;
 }

 rem_data = size;
 *keylen = size;
 while (rem_data > 0) {
 uint16_t chunk = (rem_data > BINARY_WRITE_MAX_LEN) ? BINARY_WRITE_MAX_LEN :
↪ rem_data;
 rem_data = rem_data - chunk;
 max_buffer = chunk;

 signatureLen = attst_data->data[0].signatureLen;
 attst_data->data[0].timeStampLen = sizeof(SE05x_TimeStamp_t);
 status = Se05x_API_ReadObject_W_Attest(&keyStore->
↪ session->s_ctx,
 keyObject->keyId,
 offset,

```

(continues on next page)

(continued from previous page)

```

 chunk,
 attestID,
 attestAlgo,
 random_attst,
 randomLen_attst,
 (key + offset),
 &max_buffer,
 attst_data->data[0].attribute,
 &(attst_data->data[0].attributeLen),
 &(attst_data->data[0].timeStamp),
 attst_data->data[0].outrandom,
 &(attst_data->data[0].outrandomLen),
 attst_data->data[0].chipId,
 &(attst_data->data[0].chipIdLen),
 attst_data->data[0].signature,
 &signatureLen);

attst_data->data[0].signatureLen -= signatureLen;
attst_data->valid_number = 1;

ENSURE_OR_GO_CLEANUP(status == SM_OK);

/* Perform verification operation here on the following data
 * (key + offset) +
 * attst_data->data[0].attribute +
 * attst_data->data[0].timestamp +
 * attst_data->data[0].outrandom +
 * attst_data->data[0].chipId
 * with signature
 * attst_data->data[0].signature
 *
 * We perform signature verification on host.
 * First we digest the data then pass it to verify API
 */

memcpy(plainData, (key + offset), max_buffer);
memcpy(plainData + max_buffer, attst_data->data[0].attribute, attst_data->
↪data[0].attributeLen);
memcpy(plainData + max_buffer + attst_data->data[0].attributeLen,
 &(attst_data->data[0].timeStamp),
 attst_data->data[0].timeStampLen);
memcpy(plainData + max_buffer + attst_data->data[0].attributeLen + attst_data->
↪data[0].timeStampLen,
 attst_data->data[0].outrandom,
 attst_data->data[0].outrandomLen);
memcpy(plainData + max_buffer + attst_data->data[0].attributeLen + attst_data->
↪data[0].timeStampLen +
 attst_data->data[0].outrandomLen,
 attst_data->data[0].chipId,
 attst_data->data[0].chipIdLen);
plainDataLen = max_buffer + attst_data->data[0].attributeLen + attst_data->
↪data[0].timeStampLen +
 attst_data->data[0].outrandomLen + attst_data->data[0].
↪chipIdLen;

sss_status = sss_digest_context_init(&digest_ctx, &pCtx->host_session, digest_
↪algorithm, kMode_SSS_Digest);

```

(continues on next page)

(continued from previous page)

```

 ENSURE_OR_GO_CLEANUP(sss_status == kStatus_SSS_Success);

 sss_status = sss_digest_one_go(&digest_ctx, plainData, plainDataLen, digest, &
↪digestLen);
 ENSURE_OR_GO_CLEANUP(sss_status == kStatus_SSS_Success);

 sss_digest_context_free(&digest_ctx);

 /* Verify signature */
 sss_asymmetric_context_init(
 &verify_ctx, &pCtx->host_session, &verification_object, algorithm, kMode_
↪SSS_Verify);

 sss_status =
 sss_asymmetric_verify_digest(&verify_ctx, digest, digestLen, attst_data->
↪data[0].signature, signatureLen);

 sss_asymmetric_context_free(&verify_ctx);

 ENSURE_OR_GO_CLEANUP(sss_status == kStatus_SSS_Success);

 offset = offset + chunk;
}

cleanup:
 return sss_status;
}

```

## 5.28 SE05X Transport Lock example

This demo Locks SE05x applet.

During transportation of the secure element, this lock can be enforced to avoid malicious access to the SE during transport.

**Warning:** This demo is just for reference for the usage of APIs, it should not be used for production.

See the corresponding demo [Section 5.29 SE05X Transport UnLock example](#)

Both the lock and unlock demo use same key and common logic from `se05x_TransportAuth.c`

## 5.29 SE05X Transport UnLock example

This demo UnLocks SE05x applet, after it is locked by [Section 5.28 SE05X Transport Lock example](#)

**Warning:** This demo is just for reference for the usage of APIs, it should not be used for production.

See the corresponding demo [Section 5.28 SE05X Transport Lock example](#)

Both the lock and unlock demo use same key and common logic from `se05x_TransportAuth.c`



## 5.30 SE05X Timestamp

The timestamp is a 12-byte value. The most significant 4 bytes are persistent and the least significant 8 bytes are transient. The persistent value is incremented at every session open and the transient 8 bytes are incremented every 100ms and reset at every session open.

So every time the timestamp is read, the newer value will always be greater than the previous value.

This example demonstrates that the timestamp value is incremented internally. Timestamp is used by the applet during attestation. We initially read the timestamp value and then perform attestation operation. We compare the initial timestamp with the timestamp received during attestation. The newer value must be greater than the older value.

### 5.30.1 Building

Build the project with the following configuration.

- Applet=SE05X\_C
- Project = se05x\_TimeStamp

### 5.30.2 Running

On successful execution, you would be able to see the timestamp printed out:

```
App :INFO :timestamp.ts (Len=12)
 00 00 00 08 00 00 00 00 00 1F BD 00
App :INFO :new_timestamp.ts (Len=12)
 00 00 00 08 00 00 00 00 00 20 B7 00
App :INFO :ex_sss Finished
```

As you can see in the above logs, the timestamp value received during attestation operation is larger than the previous value.

## 5.31 MIFARE DESFire EV2 : Prepare SE050

### 5.31.1 Prerequisites

- Bring Up Hardware. (Refer [Development Platforms](#))
- Connect RC663 to your microncontroller. (Refer [mifarekdf-rc663](#))

### 5.31.2 About the Example

This is an example project for provisioning the SE050 for running other SE050 MIFARE DESFire EV2 examples. This example Creates and sets 2 SE050 secure objects for further use by other examples.

It uses the following APIs and data types:

- `sss_key_store_set_key()`

### 5.31.3 The two AES key storage for NFC application n SE050

We create two AES Keys in SE050

This example calls the - `InitialSetupSe050()`: which inits, allocates and sets 2 AES keys.

The keyIDs are as below

```
#define MFD FEV2_KEYID (EX_SSS_OBJID_DEMO_MFDF_START)
#define MFD FEV2_CHANGED_KEYID (EX_SSS_OBJID_DEMO_MFDF_START +1)
```

```
#define EX_SSS_OBJID_DEMO_MFDF_START 0x7D5DF000u
```

#### Key values

The key that is provisioned into SE050 is called the oldKey and newKey and it takes the values as below.

```
const uint8_t oldKeyValue[KEY_BIT_LEN / 8] = { 0x00, 0x00, 0x00, 0x00,
 0x00, 0x00, 0x00, 0x00,
 0x00, 0x00, 0x00, 0x00,
 0x00, 0x00, 0x00, 0x00 };

const uint8_t newKeyValue[KEY_BIT_LEN / 8] = { 0x00, 0x00, 0x00, 0x00,
 0x00, 0x00, 0x00, 0x00,
 0x00, 0x00, 0x00, 0x00,
 0x00, 0x00, 0x00, 0x01 };
```

### 5.31.4 Running the Demo

- 1) Either press the reset button on your board or launch the debugger in your IDE to begin running the demo

If everything is setup correctly the output would be as follows:

```
sss:INFO :atr (Len=35)
 00 A0 00 00 03 96 04 03 E8 00 FE 02 0B 03 E8 08
 01 00 00 00 00 64 00 00 0A 4A 43 4F 50 34 20 41
 54 50 4F
sss:WARN :Communication channel is Plain.
sss:WARN :!!!Not recommended for production use.!!!
App:INFO :SE050 prepared successfully for MIFARE DESFire EV2 examples
App:INFO :ex_sss Finished
```

## 5.32 MIFARE DESFire EV2 : Prepare MFD FEV2

**Warning:** To run this example, you would need the nxpnfcrdlib component for which a Non-Disclosure Agreement(NDA) needs to be signed. Please contact your FAE for additional details.

### 5.32.1 Prerequisites

- Bring Up Hardware. (Refer *Development Platforms*)

- Connect RC663 to your microncontroller. (Refer [mifarekdf-rc663](#))

### 5.32.2 About the Example

This is an example project for preparing the MIFARE DESFire EV2 card for running other SE050 MIFARE DESFire EV2 examples. This example formats the card creates app, keys and file for running the other examples.

This does not make use of any SE05X APIs. It make use of nxpNfcrdLib apis for - formatting the card - Creating an AFC application - Selecting the application - Creating a value file

### 5.32.3 Creation of application

creation of application with application ID happens in - `phEx_Personalize_AFCApp()`:

The application ID is

```
uint8_t bAfcApp[3] = {0x11, 0x22, 0x33};
```

### 5.32.4 AES Keys provisioned in card

We create One key in the card, This key is set when we are creating the transaction mac file. The key value is

```
uint8_t bTMKey[16] = { 0x00, 0x00, 0x00, 0x00,
 0x00, 0x00, 0x00, 0x00,
 0x00, 0x00, 0x00, 0x00,
 0x00, 0x00, 0x00, 0x00 };
```

### 5.32.5 Running the Demo

- 1) Either press the reset button on your board or launch the debugger in your IDE to begin running the demo

If everything is setup correctly the output would be as follows:

```
App:INFO :PlugAndTrust_v02.09.00_20190607
sss:INFO :atr (Len=35)
 00 A0 00 00 03 96 04 03 E8 00 FE 02 0B 03 E8 08
 01 00 00 00 00 64 00 00 0A 4A 43 4F 50 34 20 41
 54 50 4F
sss:WARN :Communication channel is Plain.
sss:WARN :!!!Not recommended for production use.!!!
Tx> 26
Rx< 04 03
Tx> 93 20
Rx< 08 4B 3F 8A F6
Tx> 93 70 08 4B 3F 8A F6
Rx< 20
UID after L3 Activation 08 4B 3F 8A 00 00 00 00 00 00
Tx> E0 80
Rx< 06 75 77 81 02 80
ATS after L4 Activation 06 75 77 81 02 80
Performing Pre Personalization
Tx> 0A 00
Tx> 0A 00
```

(continues on next page)

(continued from previous page)

```

Rx< 0A 00 AF CB 25 7E C2 10 CE 82 78
Tx> 0B 00
Tx> AF 04 B9 B6 CE 0D 73 DE 3F 4D 9F 70 01 F0 12 53 B7
Rx< 0B 00 00 88 CC 75 06 D0 EA FE F2
Tx> 0A 00
Tx> FC
Rx< FA 00 01
Tx> FA 00 01
Rx< 0A 00 00
Formating the card Successful
Tx> 0B 00
Tx> 5C 01
Tx> 72 38 8B 51 41 68 C4 08 BD C3 F4 15 C9 BF D1 56 CA EC FD 6C 18 AC 10 9B
Tx> 11 79 59 17 C3 0D 57 B3
Rx< 0B 00 00
Tx> 0A 00
Tx> 5C 00
Tx> 1C DE 04 43 CC 47 C9 9B
Rx< 0A 00 00
Tx> 0B 00
Tx> 51
Rx< 0B 00 00 49 27 EF 57 76 83 6C 8F A3 7B D7 DF 6E E1 6D 0E
bCardUid[0] = 0x4
bCardUid[1] = 0x40
bCardUid[2] = 0x40
bCardUid[3] = 0x42
bCardUid[4] = 0x4d
bCardUid[5] = 0x4d
bCardUid[6] = 0x80
bCardUid[7] = 0x0
bCardUid[8] = 0x0
bCardUid[9] = 0x0
Tx> 0A 00
Tx> CA 11 22 33 0F B6 01 01 02 10 00 AA 00 41 46 43 41 50 50 4C 49 43 41 54 49 4F 4E
Rx< 0A 00 00
Create AFC Application Successful
Tx> 0B 00
Tx> 5A 11 22 33
Rx< 0B 00 00
Select the AFC Application Successful
phEx_Create_ValueFile...
Tx> 0A 00
Tx> CE 05 03 F0 0F 02 00
Rx< 0A 00 00
 create Transaction MAC File Successful
Tx> 0B 00
Tx> CC 03 03 00 00 14 00 00 00 F4 01 00 00 14 00 00 00 02
Rx< 0B 00 00
 create Value File Successful
Tx> 0A 00
Tx> CD 1F 56 78 00 E0 00 FF 00 00
Rx< 0A 00 00
***** Creating standard data file SUCCESS!!*****
App:INFO :ex_sss Finished

```

## 5.33 MIFARE DESFire EV2 : Authentication

**Warning:** To run this example, you would need the nxpnfcrdlib component for which a Non-Disclosure Agreement(NDA) needs to be signed. Please contact your FAE for additional details.

### 5.33.1 Prerequisites

- *MIFARE DESFire EV2 : Prepare MFDFEV2* must have been executed, so that the MIFARE DESFire EV2 card has the required credentials.
- *MIFARE DESFire EV2 : Prepare SE050* must have been executed, so that the Secure element has the required credentials.
- Bring Up Hardware. (Refer *Development Platforms*)
- Connect RC663 to your microcontroller. (Refer *mifarekdf-rc663*)

### 5.33.2 About the Example

This project is an example demonstrating the Mifare Desfire EV2 authentication using Seo50. After authentication it performs encrypted communication with the desfire EV2 card.

It uses the following APIs and data types:

- *Se05x\_API\_DFAuthenticateFirstPart1()*
- *Se05x\_API\_DFAuthenticateNonFirstPart1()*
- *Se05x\_API\_DFAuthenticateFirstPart2()*
- *Se05x\_API\_DFAuthenticateNonFirstPart2()*
- *Se05x\_API\_DFKillAuthentication()*

### 5.33.3 Running the Demo

- 1) Either press the reset button on your board or launch the debugger in your IDE to begin running the demo

If everything is setup correctly the output would be as follows:

```
App:INFO :PlugAndTrust_v02.09.00_20190607
 sss:INFO :atr (Len=35)
 00 A0 00 00 03 96 04 03 E8 00 FE 02
↪ 0B 03 E8 08 01 00 00 00 00 64 00 00 0A 4A 43 4F
↪ 50 34 20 41 54 50 4F
 sss:WARN :Communication channel is Plain.
 sss:WARN :!!!Not recommended for production use.!!!
Tx> 26
Rx< 04 03
Tx> 93 20
Rx< 08 7D 6B 96 88
Tx> 93 70 08 7D 6B 96 88
Rx< 20
```

(continues on next page)

(continued from previous page)

```

UID after L3 Activation 08 7D 6B 96 00 00 00 00 00 00
Tx> E0 80
Rx< 06 75 77 81 02 80
ATS after L4 Activation 06 75 77 81 02 80
Tx> 0A 00
Tx> 5A 11 22 33
Rx< 0A 00 00
 App:INFO :Select the AFC Application Successful

 App:INFO :attempting to authenticate with cardkey = 0 and Se0Obj ID =
 ↪2103308288
Tx> 0B 00
Tx> 71 00 06 00 00 00 00 00 00
Rx< 0B 00 AF 35 58 48 3C 62 54 DE 34 F5 78 03 24 14 53 13 7C
 App:INFO :
 CARD =====> SE050 16-byte Ek(RndB) =
 (Len=16)
 35 58 48 3C 62 54 DE 34 F5 78 03 24
 ↪ 14 53 13 7C
 App:INFO :
 CARD <===== SE050 E(Kx, RandA || RandB') =
 (Len=32)
 1B 45 77 91 64 D2 57 B5 20 40 95 FD
 ↪ 40 B8 D8 D4 74 C4 51 5D 5F A4 07 2C
 ↪ EF 15 4F 62 0D 6F 8F 6B
 Tx> 0A 00
Tx> AF 1B 45 77 91 64 D2 57 B5 20 40 95 FD 40 B8 D8 D4 74 C4 51 5D 5F A4 07 2C 0D 6F
 ↪8F 6B EF 15 4F 62
Rx< 0A 00 00 78 4A 85 D0 4A 30 F5 B4 48 B6 EC 98 1F F8 43 62 BE 6E B9 03 2C 24 EF 19
 ↪5D A4 95 2B D8 B7 C6 8A
 App:INFO :
 CARD =====> SE050 32-byte E(Kx, TI||RandA'||PDCap2||PCDcap2) =
 (Len=32)
 78 4A 85 D0 4A 30 F5 B4 48 B6 EC 98
 ↪ 1F F8 43 62 BE 6E B9 03 2C 24 EF 19
 ↪ D8 B7 C6 8A 5D A4 95 2B
 App:INFO :
 CARD <===== SE050 E(Kx, RandA || RandB') =
 (Len=12)
 00 00 00 00 00 00 00 00 00 00 00 00
 App:INFO :Dumped Session Key is (Len=16)
 41 39 2E 7D 09 3E 19 DD 17 8B 0E 73
 ↪ E5 1E 12 B9
 App:INFO :Dumped Session Mac is (Len=16)
 9D 15 63 F3 2D 66 1F 4B D6 3A 16 15
 ↪ E1 3F A6 97
 App:INFO :Dumped TI is (Len=4)
 95 E5 18 5D
 App:INFO :pDataParams->wCmdCtr=0
 App:INFO : EV2 First Authenticate Successful

Tx> 0B 00
Tx> 77 00
Rx< 0B 00 AF 50 5C 0C 05 50 7D BC 3A 8E 6B 6C 88 B1 E2 A5 B0
 App:INFO :

```

(continues on next page)

(continued from previous page)

```

CARD =====> SE050 16-byte Ek(RndB) =
(Len=16)
 50 5C 0C 05 50 7D BC 3A 8E 6B 6C 88
↪ B1 E2 A5 B0
App:INFO :
CARD <===== SE050 E(Kx, RandA || RandB') =
(Len=32)
 11 F3 C6 A4 44 EF A1 85 2F 37 AF 64
↪ 8F 6F 51 12 6E BD E7 B5 5B E6 B1 38 9A 4F 7D A7
↪ 8F 65 40 20
Tx> 0A 00
Tx> AF 11 F3 C6 A4 44 EF A1 85 2F 37 AF 64 8F 6F 51 12 6E BD E7 B5 5B E6 B1 38 9A 4F
↪ 7D A7 8F 65 40 20
Rx< 0A 00 00 19 D9 C9 E8 4E 52 DE 9E C5 41 02 C7 80 68 76 A1
App:INFO :
CARD =====> SE050 32-byte E(Kx, TI||RndA'||PDCap2||PCDcap2) =
(Len=16)
 19 D9 C9 E8 4E 52 DE 9E C5 41 02 C7
↪ 80 68 76 A1
App:INFO :Dumped Session Key is (Len=16)
 12 EF F8 40 59 61 99 DA B8 C4 08 3D
↪ 90 8B 73 51
App:INFO :Dumped Session Mac is (Len=16)
 FF 23 11 DC 17 A9 7A 0B 5C D0 45 AF
↪ 86 F1 87 3E
App:INFO :Dumped TI is (Len=4)
 95 E5 18 5D
App:INFO :pDataParams->wCmdCtr=0
App:INFO : EV2 Following Authenticate Successful

App:INFO :Authenticated with cardkey = 0 and Se0Obj ID = 2103308288
Tx> 0B 00
Tx> 51 D8 49 E7 5E F0 9B 7A 0F
Rx< 0B 00 00 B0 1F 31 B1 91 54 B6 14 42 72 B7 CD BA 24 13 66 63 CF D9 A4 F0 1B B4 00
App:INFO :CARD UID is as below (Len=7)
 04 40 40 42 4D 4D 80
phEx_Use_ValueFile...
Performing Accreditation in AFC App....
Performing Accreditation in AFC App Successful
(Plain Communicatioon)Trying to Get the Current Value. Plain Communicatioon
Tx> 0A 00
Tx> 6C 03
Rx< 0A 00 00 14 00 00 00
Getting current value Successful
(Enc Communication using session Key)Trying to Add money to the account
Tx> 0B 00
Tx> 0C 03
Tx> 78 E3 12 67 90 96 56 EA 91 88 6A B8 08 46 A2 14 49 37 CE 63 E2 27 7C 1B
Rx< 0B 00 00 E2 CE 37 F1 09 D8 E1 EA
Add money to the account successful
Tx> 0A 00
Tx> C7 01 8A 66 56 AC 96 BA B6 2E
Rx< 0A 00 00 01 00 00 00 71 76 A5 CC 92 DA 07 A4 C2 23 23 28 FC EF 11 C7
Tx> 0B 00
Tx> 6C 03
Rx< 0B 00 00 17 00 00 00

```

(continues on next page)

(continued from previous page)

```
The amount in your account After credit is 0 0 0 17
Accreditation DONE!
App:INFO : Auth session is reset in software
App:INFO : Auth session is killed in SE
App:INFO :ex_sss Finished
```

## 5.34 MIFARE DESFire EV2 : Change Key

**Warning:** To run this example, you would need the nxpnfcrdlib component for which a Non-Disclosure Agreement(NDA) needs to be signed. Please contact your FAE for additional details.

### 5.34.1 Prerequisites

- *MIFARE DESFire EV2 : Prepare MFDFEV2* must have been executed, so that the MIFARE DESFire EV2 card has the required credentials.
- *MIFARE DESFire EV2 : Prepare SE050* must have been executed, so that the Secure element has the required credentials.
- Bring Up Hardware. (Refer *Development Platforms*)
- Connect RC663 to your microncontroller. (Refer *mifarekdf-rc663*)

### 5.34.2 About the Example

This project demonstrates the Mifare Desfire EV2 ChangeKeyEv2 using Se050. After Changing Keys, it performs encrypted communication with the desfire EV2 card using the changed key. If enabled, It also reverts back the changed key.

It uses the following APIs and data types:

- *Se05x\_API\_DFChangeKeyPart1()*
- *Se05x\_API\_DFChangeKeyPart2()*
- *Se05x\_API\_DFAuthenticateFirstPart1()*
- *Se05x\_API\_DFAuthenticateNonFirstPart1()*
- *Se05x\_API\_DFAuthenticateFirstPart2()*
- *Se05x\_API\_DFAuthenticateNonFirstPart2()*
- *Se05x\_API\_DFKillAuthentication()*



### 5.34.3 Running the Demo

1) Either press the reset button on your board or launch the debugger in your IDE to begin running the demo

If everything is setup correctly the output would be as follows:

```

App:INFO :PlugAndTrust_v02.09.00_20190607
sss:INFO :atr (Len=35)
00 A0 00 00 03 96 04 03 E8 00 FE 02 0B 03 E8 08
01 00 00 00 00 64 00 00 0A 4A 43 4F 50 34 20 41
54 50 4F
sss:WARN :Communication channel is Plain.
sss:WARN :!!!Not recommended for production use.!!!
Tx> 26
Rx< 04 03
Tx> 93 20
Rx< 08 00 FD 08 FD
Tx> 93 70 08 00 FD 08 FD
Rx< 20
UID after L3 Activation 08 00 FD 08 00 00 00 00 00
Tx> E0 80
Rx< 06 75 77 81 02 80
ATS after L4 Activation 06 75 77 81 02 80
Tx> 0A 00
Tx> 5A 11 22 33
Rx< 0A 00 00
App:INFO :Select the AFC Application Successful

App:INFO :attempting to authenticate with cardkey = 0 and Se0Obj ID =
↪2103308288
Tx> 0B 00
Tx> 71 00 06 00 00 00 00 00
Rx< 0B 00 AF B4 BF 4B CE F5 D6 B4 A8 78 7B 67 48 E2 9A 69 E2
App:INFO :
CARD <====> SE050 16-byte Ek(RndB) =
(Len=16)
B4 BF 4B CE F5 D6 B4 A8 78 7B 67 48 E2 9A 69 E2
App:INFO :
CARD <==== SE050 E(Kx, RandA || RandB') =
(Len=32)
EF A9 8E BA 7A F6 29 E3 09 26 5C 2D 15 09 72 C1
5B D3 8B FA 35 97 D4 74 C0 CC E9 3D 2C FC 0F B0
Tx> 0A 00
Tx> AF EF A9 8E BA 7A F6 29 E3 09 26 5C 2D 15 09 72 C1 5B D3 8B FA 35 97 D4 74 C0 CC
↪E9 3D 2C FC 0F B0
Rx< 0A 00 00 AA 21 3F 9C E2 93 74 37 96 7A 4B 2E E5 25 D0 6C A6 D2 95 03 56 05 51 E6
↪D8 76 BF 26 13 1A 54 E9
App:INFO :
CARD <====> SE050 32-byte E(Kx, TI||RandA'||PDCap2||PCDcap2) =
(Len=32)
AA 21 3F 9C E2 93 74 37 96 7A 4B 2E E5 25 D0 6C
A6 D2 95 03 56 05 51 E6 D8 76 BF 26 13 1A 54 E9
App:INFO :
CARD <==== SE050 E(Kx, RandA || RandB') =
(Len=12)
00 00 00 00 00 00 00 00 00 00 00 00
App:INFO :Dumped Session Key is (Len=16)
C0 F5 21 9F 2C 30 E6 A8 59 F8 91 3E F5 8B D5 E8

```

(continues on next page)

(continued from previous page)

```

App:INFO :Dumped Session Mac is (Len=16)
0B 69 27 1D 5F 13 EA 2D 5B 7A 96 7A 52 D7 78 26
App:INFO :Dumped TI is (Len=4)
A2 27 15 FA
App:INFO :pDataParams->wCmdCtr=0
App:INFO : EV2 First Authenticate Successful

Tx> 0B 00
Tx> 77 00
Rx< 0B 00 AF AA 90 E0 4D EA CA 82 06 CF BD 71 EC CC D4 13 EA
App:INFO :
CARD ==> SE050 16-byte Ek(RndB) =
(Len=16)
AA 90 E0 4D EA CA 82 06 CF BD 71 EC CC D4 13 EA
App:INFO :
CARD <===== SE050 E(Kx, RandA || RandB') =
(Len=32)
2D AC 96 1B 3D 9D 1D A9 FB A5 50 C3 D9 77 B1 AD
6B E8 A1 1A 3A B9 70 59 4D DC 97 0E 43 FF 4E 1B
Tx> 0A 00
Tx> AF 2D AC 96 1B 3D 9D 1D A9 FB A5 50 C3 D9 77 B1 AD 6B E8 A1 1A 3A B9 70 59 4D DC
↪97 0E 43 FF 4E 1B
Rx< 0A 00 00 C2 EA 3C CD 57 5A B5 BA 28 E1 ED 9F 5A 25 43 54
App:INFO :
CARD ==> SE050 32-byte E(Kx, TI||RandA'||PDCap2||PCDcap2) =
(Len=16)
C2 EA 3C CD 57 5A B5 BA 28 E1 ED 9F 5A 25 43 54
App:INFO :Dumped Session Key is (Len=16)
63 BD 6A 50 20 21 B7 75 37 60 E2 20 FB 8C AE 16
App:INFO :Dumped Session Mac is (Len=16)
23 8F 24 BE C0 3B BA 91 BB CB 4C 5B 35 AE F7 2B
App:INFO :Dumped TI is (Len=4)
A2 27 15 FA
App:INFO :pDataParams->wCmdCtr=0
App:INFO : EV2 Following Authenticate Successful

App:INFO :Authenticated with cardkey = 0 and Se0Obj ID = 2103308288
App:INFO :attempting to change cardkey = 2 from Old Se050ObjID= 2103308288 to
↪new Se050ObjID= 2103308289
Tx> 0B 00
Tx> C6 00 02 1A 76 EC 43 C7 84 68 A3 9B 6F 48 89 3F 28 0E B3 84 88 1A B8 04 45 0C 93
↪98 B3 EF E7 69 81 BC A5 00 C9 71 35 A0 DF 8B D8
Rx< 0B 00 00 05 5E 90 58 DB D5 33 95
App:INFO : Change Key for card key 2 is Successful to Se050ObjID= 2103308289

App:INFO :Checking that the previous auth session is still valid by trying an
↪encrypted communication
Tx> 0A 00
Tx> 51 57 8B 91 0C E6 28 06 A0
Rx< 0A 00 00 5D 03 C2 BB 05 1D 51 D1 91 F9 88 01 E4 F9 27 D2 69 F3 B4 67 D0 FA 65 D4
App:INFO :CARD UID is as below (Len=7)
04 40 40 42 4D 4D 80
App:INFO :Previous auth session is still valid
App:INFO :Auth with the changed cardkey 2
Tx> 0B 00
Tx> 5A 11 22 33
Rx< 0B 00 00

```

(continues on next page)

(continued from previous page)

```

App:INFO :Select the AFC Application Successful

App:INFO :attempting to authenticate with cardkey = 2 and Se0Obj ID =
↪2103308289
Tx> 0A 00
Tx> 71 02 06 00 00 00 00 00
Rx< 0A 00 AF E5 9E 5D E4 DB 0F 51 D7 79 46 18 7A CB 06 D5 0E
App:INFO :
CARD ==> SE050 16-byte Ek(RndB) =
(Len=16)
E5 9E 5D E4 DB 0F 51 D7 79 46 18 7A CB 06 D5 0E
App:INFO :
CARD <===== SE050 E(Kx, RandA || RandB') =
(Len=32)
D6 C5 48 6C C8 60 58 CC D5 8A 6D 80 3A DD DF 32
A3 CA FD AF 8A BA 71 B6 F6 7C 51 71 AA D6 46 AA
Tx> 0B 00
Tx> AF D6 C5 48 6C C8 60 58 CC D5 8A 6D 80 3A DD DF 32 A3 CA FD AF 8A BA 71 B6 F6 7C
↪51 71 AA D6 46 AA
Rx< 0B 00 00 8A 86 8C AF ED F4 DB 56 68 81 71 1E 96 AD EF E6 5A A0 DD DA C5 BB 2E 9F
↪BE 0F 7F 39 1F 0C 4B BF
App:INFO :
CARD ==> SE050 32-byte E(Kx, TI||RandA'||PCDcap2||PCDcap2) =
(Len=32)
8A 86 8C AF ED F4 DB 56 68 81 71 1E 96 AD EF E6
5A A0 DD DA C5 BB 2E 9F BE 0F 7F 39 1F 0C 4B BF
App:INFO :
CARD <===== SE050 E(Kx, RandA || RandB') =
(Len=12)
00 00 00 00 00 00 00 00 00 00 00 00
App:INFO :Dumped Session Key is (Len=16)
6E B1 6E 85 61 F3 B6 23 6D CD 72 66 35 A9 30 0A
App:INFO :Dumped Session Mac is (Len=16)
31 2D 73 1C F6 90 66 AA 60 B4 C0 34 D5 03 54 6D
App:INFO :Dumped TI is (Len=4)
36 D6 43 CB
App:INFO :pDataParams->wCmdCtr=0
App:INFO : EV2 First Authenticate Successful

Tx> 0A 00
Tx> 77 02
Rx< 0A 00 AF 68 01 92 38 38 81 DB EE EB 9C 49 05 EA AD 1C B5
App:INFO :
CARD ==> SE050 16-byte Ek(RndB) =
(Len=16)
68 01 92 38 38 81 DB EE EB 9C 49 05 EA AD 1C B5
App:INFO :
CARD <===== SE050 E(Kx, RandA || RandB') =
(Len=32)
01 A7 15 73 7F C3 34 4D 6C C7 43 9B BB 1C 44 DE
28 78 DE 4B 69 43 53 02 C9 85 A6 21 6F F5 73 E0
Tx> 0B 00
Tx> AF 01 A7 15 73 7F C3 34 4D 6C C7 43 9B BB 1C 44 DE 28 78 DE 4B 69 43 53 02 C9 85
↪A6 21 6F F5 73 E0
Rx< 0B 00 00 8F 15 A7 DB DE 7F 4B D7 AD C4 B6 4C E4 BD 37 BD
App:INFO :
CARD ==> SE050 32-byte E(Kx, TI||RandA'||PCDcap2||PCDcap2) =

```

(continues on next page)

(continued from previous page)

```
(Len=16)
 8F 15 A7 DB DE 7F 4B D7 AD C4 B6 4C E4 BD 37 BD
App:INFO :Dumped Session Key is (Len=16)
BD 61 A3 45 2F 91 FA E4 87 7A 0A 6C 33 B7 29 B8
App:INFO :Dumped Session Mac is (Len=16)
87 87 14 42 F4 16 E5 74 F0 12 5F AA B8 2C 70 86
App:INFO :Dumped TI is (Len=4)
36 D6 43 CB
App:INFO :pDataParams->wCmdCtr=0
App:INFO : EV2 Following Authenticate Successful

App:INFO :Authenticated with cardkey = 2 and Se0Obj ID = 2103308289
App:INFO :Checking that the auth session with changed key is valid by trying
↳an encrypted communication
Tx> 0A 00
Tx> 51 DA FA 27 73 68 30 43 16
Rx< 0A 00 00 F3 40 C6 9E 11 23 4C A5 22 A7 35 30 D2 DB CD B3 D4 3D 89 66 CA 3D 57 7D
App:INFO :CARD UID is as below (Len=7)
04 40 40 42 4D 4D 80
App:INFO :Encrypted communication with changed key successful
App:INFO :Reverting the changed cardkey 2
Tx> 0B 00
Tx> 5A 11 22 33
Rx< 0B 00 00
App:INFO :Select the AFC Application Successful

App:INFO :attempting to authenticate with cardkey = 0 and Se0Obj ID =
↳2103308288
Tx> 0A 00
Tx> 71 00 06 00 00 00 00 00
Rx< 0A 00 AF 41 EB 4E 85 BB A2 31 85 F6 8F 4B 7D FD 55 02 BC
App:INFO :
CARD >====> SE050 16-byte Ek(RndB) =
(Len=16)
41 EB 4E 85 BB A2 31 85 F6 8F 4B 7D FD 55 02 BC
App:INFO :
CARD <===== SE050 E(Kx, RandA || RandB') =
(Len=32)
7F 08 80 7A AD 97 69 12 95 C3 F5 5C 72 03 D6 D4
F6 22 D4 ED 43 44 72 E2 C2 99 82 52 57 1D 80 57
Tx> 0B 00
Tx> AF 7F 08 80 7A AD 97 69 12 95 C3 F5 5C 72 03 D6 D4 F6 22 D4 ED 43 44 72 E2 C2 99
↳82 52 57 1D 80 57
Rx< 0B 00 00 E3 73 4F F3 50 65 30 02 2F B0 B6 0A 49 57 43 E0 75 49 3C 5D C1 DE B4 78
↳84 8D 00 9A 67 85 33 EA
App:INFO :
CARD >====> SE050 32-byte E(Kx, TI||RandA'||PDCap2||PCDcap2) =
(Len=32)
E3 73 4F F3 50 65 30 02 2F B0 B6 0A 49 57 43 E0
75 49 3C 5D C1 DE B4 78 84 8D 00 9A 67 85 33 EA
App:INFO :
CARD <===== SE050 E(Kx, RandA || RandB') =
(Len=12)
00 00 00 00 00 00 00 00 00 00 00 00
App:INFO :Dumped Session Key is (Len=16)
B3 44 56 9F 04 11 40 DF 5C DF 1A 24 82 E6 24 C7
App:INFO :Dumped Session Mac is (Len=16)
```

(continues on next page)

(continued from previous page)

```

16 18 AA 95 F0 E9 40 0A DA 22 FF 76 02 7B D6 8F
App:INFO :Dumped TI is (Len=4)
0D AC 1D F2
App:INFO :pDataParams->wCmdCtr=0
App:INFO : EV2 First Authenticate Successful

Tx> 0A 00
Tx> 77 00
Rx< 0A 00 AF E0 7B AA 84 C9 44 56 DE 96 72 FD 80 7C 27 31 D9
App:INFO :
CARD ==> SE050 16-byte Ek(RndB) =
(Len=16)
E0 7B AA 84 C9 44 56 DE 96 72 FD 80 7C 27 31 D9
App:INFO :
CARD <===== SE050 E(Kx, RandA || RandB') =
(Len=32)
D7 76 C8 85 E4 A1 5B 43 F0 8B F5 E5 30 FE 1C 24
EA 93 A5 AB 78 D9 82 D2 1A 59 E5 D3 25 8A CA D2
Tx> 0B 00
Tx> AF D7 76 C8 85 E4 A1 5B 43 F0 8B F5 E5 30 FE 1C 24 EA 93 A5 AB 78 D9 82 D2 1A 59
↪E5 D3 25 8A CA D2
Rx< 0B 00 00 0C A8 D5 4E FB F5 87 00 F3 2F 27 E5 4F 9A 8A 39
App:INFO :
CARD ==> SE050 32-byte E(Kx, TI||RandA'||PDCap2||PCDcap2) =
(Len=16)
0C A8 D5 4E FB F5 87 00 F3 2F 27 E5 4F 9A 8A 39
App:INFO :Dumped Session Key is (Len=16)
00 8E 4C DE 33 20 50 4C 52 23 CC BF 15 BD E2 27
App:INFO :Dumped Session Mac is (Len=16)
00 AE 0D 01 C1 4D 47 86 57 04 60 6D 3F DD A5 A0
App:INFO :Dumped TI is (Len=4)
0D AC 1D F2
App:INFO :pDataParams->wCmdCtr=0
App:INFO : EV2 Following Authenticate Successful

App:INFO :Authenticated with cardkey = 0 and Se0Obj ID = 2103308288
App:INFO :attempting to change cardkey = 2 from Old Se050ObjID= 2103308289 to
↪new Se050ObjID= 2103308288
Tx> 0A 00
Tx> C6 00 02 0B 44 52 C6 A9 EE 58 C1 9B 03 4F 64 58 DA 86 AA DE DE 68 03 8A AC F4 E2
↪8A E5 B1 F3 84 F5 87 F5 3D DC CC 01 DF D8 E7 68
Rx< 0A 00 00 9E C3 E2 37 5E 2D 40 E6
App:INFO : Change Key for card key 2 is Successful to Se050ObjID= 2103308288

App:INFO :Reverting Successful
Tx> 0B 00
Tx> 5A 11 22 33
Rx< 0B 00 00
App:INFO :Select the AFC Application Successful

App:INFO :attempting to authenticate with cardkey = 0 and Se0Obj ID =
↪2103308288
Tx> 0A 00
Tx> 71 00 06 00 00 00 00 00 00
Rx< 0A 00 AF 28 5A 92 39 D1 81 2E 77 92 07 5D C6 B2 8E 57 20
App:INFO :
CARD ==> SE050 16-byte Ek(RndB) =

```

(continues on next page)

(continued from previous page)

```
(Len=16)
 28 5A 92 39 D1 81 2E 77 92 07 5D C6 B2 8E 57 20
 App:INFO :
 CARD <===== SE050 E(Kx, RandA || RandB') =
(Len=32)
 DE E0 70 04 F2 E8 55 7C 83 D7 D0 B0 EA 4B AA 60
 F5 4F B5 AF 9E 73 A0 DD E6 EF 77 2C BD 6D 99 BE
Tx> 0B 00
Tx> AF DE E0 70 04 F2 E8 55 7C 83 D7 D0 B0 EA 4B AA 60 F5 4F B5 AF 9E 73 A0 DD E6 EF
↪77 2C BD 6D 99 BE
Rx< 0B 00 00 B7 55 91 A4 03 DA 29 3B ED 0D 63 E4 21 17 DC 86 D8 2C 62 39 4E EC B3 A0
↪31 82 64 BD 51 A1 D1 A0
 App:INFO :
 CARD =====> SE050 32-byte E(Kx, TI||RndA'||PDCap2||PCDcap2) =
(Len=32)
 B7 55 91 A4 03 DA 29 3B ED 0D 63 E4 21 17 DC 86
 D8 2C 62 39 4E EC B3 A0 31 82 64 BD 51 A1 D1 A0
 App:INFO :
 CARD <===== SE050 E(Kx, RandA || RandB') =
(Len=12)
 00 00 00 00 00 00 00 00 00 00 00 00
 App:INFO :Dumped Session Key is (Len=16)
 1C 92 1F 77 EC A4 26 FD 20 67 3C C0 63 62 20 15
 App:INFO :Dumped Session Mac is (Len=16)
 64 B2 DA 8B 05 8C 8B 24 1E AC 5E 31 CF B7 76 5D
 App:INFO :Dumped TI is (Len=4)
 40 77 24 27
 App:INFO :pDataParams->wCmdCtr=0
 App:INFO : EV2 First Authenticate Successful

Tx> 0A 00
Tx> 77 00
Rx< 0A 00 AF C8 F3 B0 76 49 C6 BF C3 43 F5 62 67 CF BF 4D CC
 App:INFO :
 CARD =====> SE050 16-byte Ek(RndB) =
(Len=16)
 C8 F3 B0 76 49 C6 BF C3 43 F5 62 67 CF BF 4D CC
 App:INFO :
 CARD <===== SE050 E(Kx, RandA || RandB') =
(Len=32)
 52 5A 01 9E B3 F2 5E 42 6F 0A 61 70 46 C2 81 54
 C4 22 67 87 99 07 49 EA B0 12 DB A3 69 24 38 4B
Tx> 0B 00
Tx> AF 52 5A 01 9E B3 F2 5E 42 6F 0A 61 70 46 C2 81 54 C4 22 67 87 99 07 49 EA B0 12
↪DB A3 69 24 38 4B
Rx< 0B 00 00 18 58 A0 DF 76 76 BB A3 F8 96 97 B7 0A DD 8C FD
 App:INFO :
 CARD =====> SE050 32-byte E(Kx, TI||RndA'||PDCap2||PCDcap2) =
(Len=16)
 18 58 A0 DF 76 76 BB A3 F8 96 97 B7 0A DD 8C FD
 App:INFO :Dumped Session Key is (Len=16)
 2A A1 EA B5 1D 7A F1 AB B7 3D 00 31 D1 26 2C 7A
 App:INFO :Dumped Session Mac is (Len=16)
 91 E9 3B CA 3D C1 53 C8 4C 13 28 28 C2 3F C9 6B
 App:INFO :Dumped TI is (Len=4)
 40 77 24 27
 App:INFO :pDataParams->wCmdCtr=0
```

(continues on next page)

(continued from previous page)

```

App:INFO : EV2 Following Authenticate Successful

App:INFO :Authenticated with cardkey = 0 and Se0Obj ID = 2103308288
App:INFO :attempting to change cardkey = 0 from Old Se050ObjID= 2103308288 to
↪new Se050ObjID= 2103308289
Tx> 0A 00
Tx> C6 00 00 A8 60 79 3E 01 31 C9 91 16 BC 07 F4 85 2D BC 7D 8B D7 CD 34 4B B4 F5 A3
↪BD 97 AD 02 89 9B 47 00 FA C8 63 0F 97 E9 55 8A
Rx< 0A 00 00
App:INFO : Change Key for card key 0 is Successful to Se050ObjID= 2103308289

App:INFO :The previous auth session is not valid anymore.
App:INFO :So.....
App:INFO :Auth with the changed cardkey 0
Tx> 0B 00
Tx> 5A 11 22 33
Rx< 0B 00 00
App:INFO :Select the AFC Application Successful

App:INFO :attempting to authenticate with cardkey = 0 and Se0Obj ID =
↪2103308289
Tx> 0A 00
Tx> 71 00 06 00 00 00 00 00 00
Rx< 0A 00 AF 96 E8 80 35 41 30 37 A7 24 8B 73 42 44 43 13 BC
App:INFO :
CARD =====> SE050 16-byte Ek(RndB) =
(Len=16)
96 E8 80 35 41 30 37 A7 24 8B 73 42 44 43 13 BC
App:INFO :
CARD <===== SE050 E(Kx, RandA || RandB') =
(Len=32)
C5 FA 74 68 B0 D8 B4 31 B6 FA 76 EF 1E 21 F4 ED
07 74 8C 1D 7C F7 5F 0F 63 24 28 7F 95 09 EF C8
Tx> 0B 00
Tx> AF C5 FA 74 68 B0 D8 B4 31 B6 FA 76 EF 1E 21 F4 ED 07 74 8C 1D 7C F7 5F 0F 63 24
↪28 7F 95 09 EF C8
Rx< 0B 00 00 F5 3D 5C 3D 95 9D A1 54 9B F8 BE 42 C0 BA EE 06 80 6F 29 91 79 89 46 FA
↪34 5C 82 C0 66 A5 CE AE
App:INFO :
CARD =====> SE050 32-byte E(Kx, TI||RandA'||PDCap2||PCDcap2) =
(Len=32)
F5 3D 5C 3D 95 9D A1 54 9B F8 BE 42 C0 BA EE 06
80 6F 29 91 79 89 46 FA 34 5C 82 C0 66 A5 CE AE
App:INFO :
CARD <===== SE050 E(Kx, RandA || RandB') =
(Len=12)
00 00 00 00 00 00 00 00 00 00 00 00
App:INFO :Dumped Session Key is (Len=16)
1B CA B6 46 6B 97 17 D0 A9 FA F4 DC 0B 01 59 CE
App:INFO :Dumped Session Mac is (Len=16)
06 B0 72 C5 8E 71 88 C0 44 51 DC 45 14 7A 42 AB
App:INFO :Dumped TI is (Len=4)
AD AB 0E F3
App:INFO :pDataParams->wCmdCtr=0
App:INFO : EV2 First Authenticate Successful

Tx> 0A 00

```

(continues on next page)

(continued from previous page)

```

Tx> 77 00
Rx< 0A 00 AF AF CA 35 37 D2 96 8B FA F5 06 CD 1E 6B D8 6E E0
 App:INFO :
 CARD >====> SE050 16-byte Ek(RndB) =
 (Len=16)
 AF CA 35 37 D2 96 8B FA F5 06 CD 1E 6B D8 6E E0
 App:INFO :
 CARD <===== SE050 E(Kx, RandA || RandB') =
 (Len=32)
 C2 72 DA 88 02 5E A0 5C 04 CC E7 86 AD 21 07 D3
 A8 74 9C 99 51 9C 62 00 00 C9 BE C9 AD FD DA 50
Tx> 0B 00
Tx> AF C2 72 DA 88 02 5E A0 5C 04 CC E7 86 AD 21 07 D3 A8 74 9C 99 51 9C 62 00 00 C9
↪BE C9 AD FD DA 50
Rx< 0B 00 00 68 C1 89 4B C1 C0 E0 20 ED 66 D1 6C D3 BE 28 99
 App:INFO :
 CARD >===== SE050 32-byte E(Kx, TI||RandA'||PDCap2||PCDcap2) =
 (Len=16)
 68 C1 89 4B C1 C0 E0 20 ED 66 D1 6C D3 BE 28 99
 App:INFO :Dumped Session Key is (Len=16)
 36 96 A4 26 4B 8F FC C3 46 A9 57 79 A7 1D 12 D3
 App:INFO :Dumped Session Mac is (Len=16)
 B8 21 B9 B6 6E 13 4F 99 E3 C7 07 89 17 09 FF 0E
 App:INFO :Dumped TI is (Len=4)
 AD AB 0E F3
 App:INFO :pDataParams->wCmdCtr=0
 App:INFO : EV2 Following Authenticate Successful

 App:INFO :Authenticated with cardkey = 0 and Se0Obj ID = 2103308289
 App:INFO :Checking that the auth session with changed key is valid by trying
↪an encrypted communication
Tx> 0A 00
Tx> 51 DE C0 9D 65 2B 8A 37 96
Rx< 0A 00 00 AD 6F 30 A2 17 04 F6 4B 0A BC 56 B1 0C 27 04 ED 87 FE 88 49 11 44 4E 94
 App:INFO :CARD UID is as below (Len=7)
 04 40 40 42 4D 4D 80
 App:INFO :Encrypted communication with changed key successful
 App:INFO :Reverting the changed cardkey 0
Tx> 0B 00
Tx> 5A 11 22 33
Rx< 0B 00 00
 App:INFO :Select the AFC Application Successful

 App:INFO :attempting to authenticate with cardkey = 0 and Se0Obj ID =
↪2103308289
Tx> 0A 00
Tx> 71 00 06 00 00 00 00 00 00
Rx< 0A 00 AF 13 41 E7 9C 31 3E 03 7F 84 46 DC 32 6A F0 81 61
 App:INFO :
 CARD >====> SE050 16-byte Ek(RndB) =
 (Len=16)
 13 41 E7 9C 31 3E 03 7F 84 46 DC 32 6A F0 81 61
 App:INFO :
 CARD <===== SE050 E(Kx, RandA || RandB') =
 (Len=32)
 2A 23 A4 C9 8F D9 42 FA 8E E2 2B 6E 5C E2 0B 63
 37 F7 D7 ED 68 33 0D 8F 84 8B 14 F7 7E 76 09 4A

```

(continues on next page)



(continued from previous page)

```

Tx> 0B 00
Tx> AF 2A 23 A4 C9 8F D9 42 FA 8E E2 2B 6E 5C E2 0B 63 37 F7 D7 ED 68 33 0D 8F 84 8B
↪14 F7 7E 76 09 4A
Rx< 0B 00 00 E6 93 12 92 02 B1 64 90 3F 85 3C 65 C0 F7 32 81 6E C8 34 3F 00 59 60 AA
↪48 DD 94 DC 48 AF C2 25
App:INFO :
CARD =====> SE050 32-byte E(Kx, TI||RndA'||PDCap2||PCDcap2) =
(Len=32)
E6 93 12 92 02 B1 64 90 3F 85 3C 65 C0 F7 32 81
6E C8 34 3F 00 59 60 AA 48 DD 94 DC 48 AF C2 25
App:INFO :
CARD <===== SE050 E(Kx, RandA || RandB') =
(Len=12)
00 00 00 00 00 00 00 00 00 00 00 00
App:INFO :Dumped Session Key is (Len=16)
22 78 6D 20 7D B1 0D 50 0D 95 B9 97 0C 0B 72 4D
App:INFO :Dumped Session Mac is (Len=16)
7A AD D3 1A 3B 3D DA 6A 7D 76 F5 64 26 A8 C4 97
App:INFO :Dumped TI is (Len=4)
48 3D F6 3C
App:INFO :pDataParams->wCmdCtr=0
App:INFO : EV2 First Authenticate Successful

Tx> 0A 00
Tx> 77 00
Rx< 0A 00 AF A6 60 13 9E 79 C2 5B 8A 12 E6 93 1B 84 BD C9 31
App:INFO :
CARD =====> SE050 16-byte Ek(RndB) =
(Len=16)
A6 60 13 9E 79 C2 5B 8A 12 E6 93 1B 84 BD C9 31
App:INFO :
CARD <===== SE050 E(Kx, RandA || RandB') =
(Len=32)
B8 95 BC 94 58 81 1D 8D 20 E4 F1 FB EE 3A 68 27
F0 BA B4 19 A6 5F 45 9A F9 A4 04 D4 3B 19 E9 6F

Tx> 0B 00
Tx> AF B8 95 BC 94 58 81 1D 8D 20 E4 F1 FB EE 3A 68 27 F0 BA B4 19 A6 5F 45 9A F9 A4
↪04 D4 3B 19 E9 6F
Rx< 0B 00 00 11 5F BE 3E 3E 3F A7 D3 3F 55 7E 44 05 C3 FC 72
App:INFO :
CARD =====> SE050 32-byte E(Kx, TI||RndA'||PDCap2||PCDcap2) =
(Len=16)
11 5F BE 3E 3E 3F A7 D3 3F 55 7E 44 05 C3 FC 72
App:INFO :Dumped Session Key is (Len=16)
8F C6 94 CF AF 83 63 3B A8 44 40 D7 18 19 39 BC
App:INFO :Dumped Session Mac is (Len=16)
C0 FF 0B A2 18 CD 33 8C 25 C5 24 E1 72 D2 F9 FB
App:INFO :Dumped TI is (Len=4)
48 3D F6 3C
App:INFO :pDataParams->wCmdCtr=0
App:INFO : EV2 Following Authenticate Successful

App:INFO :Authenticated with cardkey = 0 and Se00bj ID = 2103308289
App:INFO :attempting to change cardkey = 0 from Old Se0500bjID= 2103308289 to
↪new Se0500bjID= 2103308288
Tx> 0A 00
Tx> C6 00 00 09 C9 09 2E 8F 81 40 6C EB B8 9C 0D 49 92 B6 C7 D2 18 87 B2 D8 EA 42 79
↪D1 99 AA 6B 56 9D 01 89 47 55 A9 8B 66 2B 51 D8

```

(continues on next page)

(continued from previous page)

```
Rx< 0A 00 00
 App:INFO : Change Key for card key 0 is Successful to Se050ObjID= 2103308288

 App:INFO :Reverting Successful
 App:INFO : Auth session is reset in software
 App:INFO : Auth session is killed in SE
 App:INFO :ex_sss Finished
```

## 5.35 MIFARE DESFire EV2 : Diversified Change Key

**Warning:** To run this example, you would need the nxpnfcrdlib component for which a Non-Disclosure Agreement(NDA) needs to be signed. Please contact your FAE for additional details.

### 5.35.1 Prerequisites

- *MIFARE DESFire EV2 : Prepare MFDFEV2* must have been executed, so that the MIFARE DESFire EV2 card has the required credentials.
- *MIFARE DESFire EV2 : Prepare SE050* must have been executed, so that the Secure element has the required credentials.
- Bring Up Hardware. (Refer *Development Platforms*)
- Connect RC663 to your microncontroller. (Refer *mifarekdf-rc663*)

### 5.35.2 About the Example

This project demonstrates the Mifare Desfire EV2 Diversified ChangeKeyEv2 using Seo50. The Key is diversified using the card UID After changing Keys, it performs encrypted communication with the desfire EV2 card using the diversified changed key. If enabled, It also reverts back the changed key.

It uses the following APIs and data types:

- `Se05x_API_DFDiversifyKey()`
- `Se05x_API_DFChangeKeyPart1()`
- `Se05x_API_DFChangeKeyPart2()`
- `Se05x_API_DFAuthenticateFirstPart1()`
- `Se05x_API_DFAuthenticateNonFirstPart1()`
- `Se05x_API_DFAuthenticateFirstPart2()`
- `Se05x_API_DFAuthenticateNonFirstPart2()`
- `Se05x_API_DFKillAuthentication()`

### 5.35.3 Running the Demo

1) Either press the reset button on your board or launch the debugger in your IDE to begin running the demo

If everything is setup correctly the output would be as follows:

```

App:INFO :PlugAndTrust_v02.09.00_20190607
sss:INFO :atr (Len=35)
00 A0 00 00 03 96 04 03 E8 00 FE 02 0B 03 E8 08
01 00 00 00 00 64 00 00 0A 4A 43 4F 50 34 20 41
54 50 4F
sss:WA App:INFO :PlugAndTrust_v02.09.00_20190607
sss:INFO :atr (Len=35)
00 A0 00 00 03 96 04 03 E8 00 FE 02 0B 03 E8 08
01 00 00 00 00 64 00 00 0A 4A 43 4F 50 34 20 41
54 50 4F
sss:WARN :Communication channel is Plain.
sss:WARN :!!!Not recommended for production use.!!!
Tx> 26
Rx< 04 03
Tx> 93 20
Rx< 08 04 4D 8D CC
Tx> 93 70 08 04 4D 8D CC
Rx< 20
UID after L3 Activation 08 04 4D 8D 00 00 00 00 00 00
Tx> E0 80
Rx< 06 75 77 81 02 80
ATS after L4 Activation 06 75 77 81 02 80
App:INFO :Auth with the cardkey 0 and getting the card UID for diversification
Tx> 0A 00
Tx> 5A 11 22 33
Rx< 0A 00 00
App:INFO :Select the AFC Application Successful

App:INFO :attempting to authenticate with cardkey = 0 and Se00Obj ID =
↪2103308288
Tx> 0B 00
Tx> 71 00 06 00 00 00 00 00 00
Rx< 0B 00 AF CB 39 4E 26 F8 1E 1D CE 4B 66 2B E7 64 2D 5D 89
App:INFO :
CARD =====> SE050 16-byte Ek(RndB) =
(Len=16)
CB 39 4E 26 F8 1E 1D CE 4B 66 2B E7 64 2D 5D 89
App:INFO :
CARD <===== SE050 E(Kx, RandA || RandB') =
(Len=32)
9A 2F 8D 8C 2F 4E 1B C3 F3 D0 ED 84 36 9E DC 03
79 A5 27 2A 66 91 F0 6D EC 4F 90 C7 E5 25 EA 51
Tx> 0A 00
Tx> AF 9A 2F 8D 8C 2F 4E 1B C3 F3 D0 ED 84 36 9E DC 03 79 A5 27 2A 66 91 F0 6D EC 4F
↪90 C7 E5 25 EA 51
Rx< 0A 00 00 A1 F9 0D 31 3A 6F 32 F2 06 69 25 1A E7 4E 3F 8C 20 EF B3 14 C5 0F DD A1
↪80 E3 13 43 C1 18 14 5D
App:INFO :
CARD =====> SE050 32-byte E(Kx, TI||RandA'||PDCap2||PCDcap2) =
(Len=32)
A1 F9 0D 31 3A 6F 32 F2 06 69 25 1A E7 4E 3F 8C
20 EF B3 14 C5 0F DD A1 80 E3 13 43 C1 18 14 5D

```

(continues on next page)

(continued from previous page)

```

App:INFO :
CARD <===== SE050 E(Kx, RandA || RandB') =
(Len=12)
00 00 00 00 00 00 00 00 00 00 00 00
App:INFO :Dumped Session Key is (Len=16)
4E 55 15 1D F5 6C F5 B4 27 2E D1 4E 50 5C C7 22
App:INFO :Dumped Session Mac is (Len=16)
36 5C C9 22 F2 F7 67 7A 47 71 12 5E 79 3F D5 43
App:INFO :Dumped TI is (Len=4)
1F C1 6F CF
App:INFO :pDataParams->wCmdCtr=0
App:INFO : EV2 First Authenticate Successful

Tx> 0B 00
Tx> 77 00
Rx< 0B 00 AF 87 4A 81 2F 50 F1 44 55 5F E6 B0 31 AB AD 3B B0
App:INFO :
CARD =====> SE050 16-byte Ek(RndB) =
(Len=16)
87 4A 81 2F 50 F1 44 55 5F E6 B0 31 AB AD 3B B0
App:INFO :
CARD <===== SE050 E(Kx, RandA || RandB') =
(Len=32)
0D D4 AA 86 A4 CA 79 9B F3 9D 2E 1F C1 64 F9 3F
47 EF 29 F9 0E 6F F1 C0 1E 0F E4 4E BB 2D D0 1F
Tx> 0A 00
Tx> AF 0D D4 AA 86 A4 CA 79 9B F3 9D 2E 1F C1 64 F9 3F 47 EF 29 F9 0E 6F F1 C0 1E 0F
↪E4 4E BB 2D D0 1F
Rx< 0A 00 00 CE 33 17 C3 41 7A 8F DF 04 F9 04 E3 CB 2D 2E BC
App:INFO :
CARD =====> SE050 32-byte E(Kx, TI||RndA'||PDCap2||PCDcap2) =
(Len=16)
CE 33 17 C3 41 7A 8F DF 04 F9 04 E3 CB 2D 2E BC
App:INFO :Dumped Session Key is (Len=16)
38 54 53 76 E8 15 A5 DD 32 28 EB D5 1F 15 87 25
App:INFO :Dumped Session Mac is (Len=16)
1C A4 94 0C 63 4C E6 63 FB F3 48 A1 8D D9 34 9B
App:INFO :Dumped TI is (Len=4)
1F C1 6F CF
App:INFO :pDataParams->wCmdCtr=0
App:INFO : EV2 Following Authenticate Successful

App:INFO :Authenticated with cardkey = 0 and Se00bj ID = 2103308288
Tx> 0B 00
Tx> 51 61 14 F1 85 4C 47 DE 42
Rx< 0B 00 00 25 D5 5B 00 6B F0 5D A9 C5 15 26 DF C2 96 D3 F3 A6 6B 4C 8E 80 E1 3D 24
App:INFO :CARD UID will be used for diversification
Tx> 0A 00
Tx> 5A 11 22 33
Rx< 0A 00 00
App:INFO :Select the AFC Application Successful

App:INFO :attempting to authenticate with cardkey = 0 and Se00bj ID =
↪2103308288
Tx> 0B 00
Tx> 71 00 06 00 00 00 00 00
Rx< 0B 00 AF 6A E9 E5 F3 4C E3 58 E5 DB 1A DE 6B A7 C6 79 68

```

(continues on next page)

(continued from previous page)

```

App:INFO :
CARD ==> SE050 16-byte Ek(RndB) =
(Len=16)
 6A E9 E5 F3 4C E3 58 E5 DB 1A DE 6B A7 C6 79 68
App:INFO :
CARD <===== SE050 E(Kx, RandA || RandB') =
(Len=32)
 54 6B B9 B4 41 F0 3D E6 37 40 28 DA 61 ED 81 97
 37 50 E0 F2 86 5D E7 E5 A2 F5 1F B5 0B 1A E7 7C
Tx> 0A 00
Tx> AF 54 6B B9 B4 41 F0 3D E6 37 40 28 DA 61 ED 81 97 37 50 E0 F2 86 5D E7 E5 A2 F5
↪ 1F B5 0B 1A E7 7C
Rx< 0A 00 00 46 24 4B 15 7C 8F D5 19 56 32 6F 11 F1 77 4B 91 59 4D CB 02 26 42 49 22
↪ C0 77 8F 67 15 68 74 D4
App:INFO :
CARD ==> SE050 32-byte E(Kx, TI||RandA'||PCDcap2||PCDcap2) =
(Len=32)
 46 24 4B 15 7C 8F D5 19 56 32 6F 11 F1 77 4B 91
 59 4D CB 02 26 42 49 22 C0 77 8F 67 15 68 74 D4
App:INFO :
CARD <===== SE050 E(Kx, RandA || RandB') =
(Len=12)
 00 00 00 00 00 00 00 00 00 00 00 00
App:INFO :Dumped Session Key is (Len=16)
 73 DE FB 5D 7C 78 E3 1A BF 97 53 35 C9 2F F4 12
App:INFO :Dumped Session Mac is (Len=16)
 CB B3 14 4E D8 15 4E 7D A2 A1 F9 B4 35 ED E2 4C
App:INFO :Dumped TI is (Len=4)
 8D 19 52 6A
App:INFO :pDataParams->wCmdCtr=0
App:INFO : EV2 First Authenticate Successful

Tx> 0B 00
Tx> 77 00
Rx< 0B 00 AF 6C 35 C4 E2 0D C8 CC 44 69 7C 75 44 7B CA 3F 36
App:INFO :
CARD ==> SE050 16-byte Ek(RndB) =
(Len=16)
 6C 35 C4 E2 0D C8 CC 44 69 7C 75 44 7B CA 3F 36
App:INFO :
CARD <===== SE050 E(Kx, RandA || RandB') =
(Len=32)
 42 96 EF CB C9 41 DD B2 F8 D5 4B 36 7F 7E F1 F0
 A5 BD E7 FB 46 D1 39 EF 1F 91 61 F4 71 B0 2D BE
Tx> 0A 00
Tx> AF 42 96 EF CB C9 41 DD B2 F8 D5 4B 36 7F 7E F1 F0 A5 BD E7 FB 46 D1 39 EF 1F 91
↪ 61 F4 71 B0 2D BE
Rx< 0A 00 00 3F 69 5E 84 B8 F5 04 F9 3B C3 63 07 67 BE 07 9E
App:INFO :
CARD ==> SE050 32-byte E(Kx, TI||RandA'||PCDcap2||PCDcap2) =
(Len=16)
 3F 69 5E 84 B8 F5 04 F9 3B C3 63 07 67 BE 07 9E
App:INFO :Dumped Session Key is (Len=16)
 4D 43 63 4D 95 98 F9 92 EB F5 AA 3C 7C 42 EE 4C
App:INFO :Dumped Session Mac is (Len=16)
 10 91 B7 3A B1 B4 2C 74 A2 71 EE 4B C8 45 94 74
App:INFO :Dumped TI is (Len=4)

```

(continues on next page)

(continued from previous page)

```

8D 19 52 6A
App:INFO :pDataParams->wCmdCtr=0
App:INFO : EV2 Following Authenticate Successful

App:INFO :Authenticated with cardkey = 0 and Se0Obj ID = 2103308288
App:INFO :attempting to change cardkey = 2 from Old Se050ObjID= 2103308288 to
↪new Se050ObjID= 2103308289
Tx> 0B 00
Tx> C6 00 02 9A 7A DA 24 18 56 CB C8 65 92 52 07 AF 5C 31 19 EB AF BF 06 CA 98 41 94
↪9F EE A5 C8 60 86 D5 67 FD 22 6D 6B 75 E4 56 5B
Rx< 0B 00 00 CA 57 0A F4 56 2C F9 7F
App:INFO : Change Key for card key 2 is Successful to Se050ObjID= 2103308289

App:INFO :Checking that the previous auth session is still valid by trying an
↪encrypted communication
Tx> 0A 00
Tx> 51 CA 0E C5 B7 E8 4D BA ED
Rx< 0A 00 00 CF 11 48 4D 32 34 D9 FE BF B9 F4 24 65 7B EC BC AD 65 93 C8 03 B7 AE 95
App:INFO :CARD UID is as below (Len=7)
04 40 40 42 4D 4D 80
App:INFO :Previous auth session is still valid
App:INFO :Auth with the changed cardkey 2
Tx> 0B 00
Tx> 5A 11 22 33
Rx< 0B 00 00
App:INFO :Select the AFC Application Successful

App:INFO :attempting to authenticate with cardkey = 2 and Se0Obj ID =
↪2103308289
Tx> 0A 00
Tx> 71 02 06 00 00 00 00 00
Rx< 0A 00 AF 75 51 82 DB 46 CA A3 B5 C8 1C 3F 16 50 46 0A D4
App:INFO :
CARD =====> SE050 16-byte Ek(RndB) =
(Len=16)
75 51 82 DB 46 CA A3 B5 C8 1C 3F 16 50 46 0A D4
App:INFO :
CARD <===== SE050 E(Kx, RandA || RandB') =
(Len=32)
6F F3 1E C6 D4 B9 A6 72 DB C6 98 54 6B E8 F4 7F
38 E3 CC 2A 34 D2 51 96 B0 D7 D4 A8 A3 30 76 4B
Tx> 0B 00
Tx> AF 6F F3 1E C6 D4 B9 A6 72 DB C6 98 54 6B E8 F4 7F 38 E3 CC 2A 34 D2 51 96 B0 D7
↪D4 A8 A3 30 76 4B
Rx< 0B 00 00 40 93 D2 A1 6B DD B2 1F 6E F4 FA 75 15 28 22 62 18 FD 68 CF AF 1C EC B0
↪41 4D 08 2E E8 13 4B A1
App:INFO :
CARD =====> SE050 32-byte E(Kx, TI||RandA'||PCDcap2||PCDcap2) =
(Len=32)
40 93 D2 A1 6B DD B2 1F 6E F4 FA 75 15 28 22 62
18 FD 68 CF AF 1C EC B0 41 4D 08 2E E8 13 4B A1
App:INFO :
CARD <===== SE050 E(Kx, RandA || RandB') =
(Len=12)
00 00 00 00 00 00 00 00 00 00 00 00
App:INFO :Dumped Session Key is (Len=16)
4A 69 72 BB 62 62 06 2F 02 7E 53 C2 B8 E9 F5 ED

```

(continues on next page)

(continued from previous page)

```

App:INFO :Dumped Session Mac is (Len=16)
47 AA 13 45 A4 6C 6F 5B D0 D8 22 C4 F2 75 DD 2C
App:INFO :Dumped TI is (Len=4)
19 3D B7 F6
App:INFO :pDataParams->wCmdCtr=0
App:INFO : EV2 First Authenticate Successful

Tx> 0A 00
Tx> 77 02
Rx< 0A 00 AF C3 38 BF 4F E5 13 7E 02 2B 8E 21 34 2F 6C 7F 6D
App:INFO :
CARD =====> SE050 16-byte Ek(RndB) =
(Len=16)
C3 38 BF 4F E5 13 7E 02 2B 8E 21 34 2F 6C 7F 6D
App:INFO :
CARD <===== SE050 E(Kx, RandA || RandB') =
(Len=32)
42 76 C2 05 F8 D5 37 1D 38 9C B3 95 A9 2A 19 5D
75 81 7C 66 7A 53 E0 6D 3C 25 B7 3D FD C7 38 16
Tx> 0B 00
Tx> AF 42 76 C2 05 F8 D5 37 1D 38 9C B3 95 A9 2A 19 5D 75 81 7C 66 7A 53 E0 6D 3C 25
↪B7 3D FD C7 38 16
Rx< 0B 00 00 12 CA C1 C4 BB B7 23 74 36 D4 17 00 4B FB 44 1F
App:INFO :
CARD =====> SE050 32-byte E(Kx, TI||RandA'||PDCap2||PCDcap2) =
(Len=16)
12 CA C1 C4 BB B7 23 74 36 D4 17 00 4B FB 44 1F
App:INFO :Dumped Session Key is (Len=16)
A2 46 74 CE 9C 90 99 2F 60 04 07 FC 9C B7 E5 3A
App:INFO :Dumped Session Mac is (Len=16)
1F D2 94 A3 9F AC E6 57 51 9F C1 BC 84 B9 FA 99
App:INFO :Dumped TI is (Len=4)
19 3D B7 F6
App:INFO :pDataParams->wCmdCtr=0
App:INFO : EV2 Following Authenticate Successful

App:INFO :Authenticated with cardkey = 2 and Se0Obj ID = 2103308289
App:INFO :Checking that the auth session with changed key is valid by trying
↪an encrypted communication
Tx> 0A 00
Tx> 51 87 B0 70 A8 75 FC FA 04
Rx< 0A 00 00 0A 31 4B 0C 44 BB 70 EA 61 31 00 E3 A3 E6 4E 1C 8E 6D 70 2F 86 08 A0 CD
App:INFO :CARD UID is as below (Len=7)
04 40 40 42 4D 4D 80
App:INFO :Encrypted communication with changed key successful
App:INFO :Reverting the changed cardkey 2
Tx> 0B 00
Tx> 5A 11 22 33
Rx< 0B 00 00
App:INFO :Select the AFC Application Successful

App:INFO :attempting to authenticate with cardkey = 0 and Se0Obj ID =
↪2103308288
Tx> 0A 00
Tx> 71 00 06 00 00 00 00 00
Rx< 0A 00 AF 95 85 6E 37 62 3C C5 16 A4 C7 A8 D2 6D A5 08 09
App:INFO :

```

(continues on next page)

(continued from previous page)

```

CARD >====> SE050 16-byte Ek(RndB) =
(Len=16)
 95 85 6E 37 62 3C C5 16 A4 C7 A8 D2 6D A5 08 09
 App:INFO :
 CARD <==== SE050 E(Kx, RandA || RandB') =
(Len=32)
 9A C1 0D 08 D6 7F FC 05 3C 2E 7B 9B 5B 8B 98 36
 29 0E 57 8D A7 BF 17 61 EB 81 CD 5C 79 2D 98 7B
Tx> 0B 00
Tx> AF 9A C1 0D 08 D6 7F FC 05 3C 2E 7B 9B 5B 8B 98 36 29 0E 57 8D A7 BF 17 61 EB 81
↪CD 5C 79 2D 98 7B
Rx< 0B 00 00 CF 77 E6 A9 D0 03 A7 C5 C2 D1 26 20 17 0C 05 8F E0 65 3D AA 3C F6 5D 5C
↪A2 8E 06 97 0B E6 73 BB
 App:INFO :
 CARD >====> SE050 32-byte E(Kx, TI||RandA'||PDCap2||PCDcap2) =
(Len=32)
 CF 77 E6 A9 D0 03 A7 C5 C2 D1 26 20 17 0C 05 8F
 E0 65 3D AA 3C F6 5D 5C A2 8E 06 97 0B E6 73 BB
 App:INFO :
 CARD <==== SE050 E(Kx, RandA || RandB') =
(Len=12)
 00 00 00 00 00 00 00 00 00 00 00 00
 App:INFO :Dumped Session Key is (Len=16)
 B5 0C 47 C4 C4 86 DA EC E7 D4 C1 9C 09 92 C5 86
 App:INFO :Dumped Session Mac is (Len=16)
 C0 36 09 35 9E 9D 00 C5 CC CF 8C 0A C6 AA 84 93
 App:INFO :Dumped TI is (Len=4)
 EE 5C D1 02
 App:INFO :pDataParams->wCmdCtr=0
 App:INFO : EV2 First Authenticate Successful

Tx> 0A 00
Tx> 77 00
Rx< 0A 00 AF 47 D5 AD E6 D8 9F 13 06 2C F8 32 CE A3 68 61 3A
 App:INFO :
 CARD >====> SE050 16-byte Ek(RndB) =
(Len=16)
 47 D5 AD E6 D8 9F 13 06 2C F8 32 CE A3 68 61 3A
 App:INFO :
 CARD <==== SE050 E(Kx, RandA || RandB') =
(Len=32)
 71 83 22 6F AD FC F8 89 F5 7F 63 15 85 87 EB 29
 7A 24 30 7F C7 D1 03 27 0A 93 EE 3F 40 68 A3 1D
Tx> 0B 00
Tx> AF 71 83 22 6F AD FC F8 89 F5 7F 63 15 85 87 EB 29 7A 24 30 7F C7 D1 03 27 0A 93
↪EE 3F 40 68 A3 1D
Rx< 0B 00 00 FF 26 85 C1 A3 96 A2 92 F5 D8 D2 98 A3 56 09 44
 App:INFO :
 CARD >====> SE050 32-byte E(Kx, TI||RandA'||PDCap2||PCDcap2) =
(Len=16)
 FF 26 85 C1 A3 96 A2 92 F5 D8 D2 98 A3 56 09 44
 App:INFO :Dumped Session Key is (Len=16)
 6A E0 C5 B9 C6 4F C2 75 C9 65 49 D8 91 FC A6 78
 App:INFO :Dumped Session Mac is (Len=16)
 A3 1F 10 92 43 D9 D2 63 1E 0C 7C A3 DC 3C 26 EC
 App:INFO :Dumped TI is (Len=4)
 EE 5C D1 02

```

(continues on next page)



(continued from previous page)

```

App:INFO :pDataParams->wCmdCtr=0
App:INFO : EV2 Following Authenticate Successful

App:INFO :Authenticated with cardkey = 0 and Se0Obj ID = 2103308288
App:INFO :attempting to change cardkey = 2 from Old Se050ObjID= 2103308289 to
↪new Se050ObjID= 2103308288
Tx> 0A 00
Tx> C6 00 02 E4 9B D3 FD 14 74 15 DC B2 D9 28 3A C5 9F BD 77 8B 12 78 3E CB 0D 4C 92
↪66 84 C8 EB 91 0E BA A2 E0 6F 53 AC 51 FC 42 A1
Rx< 0A 00 00 37 FD 0F 81 21 25 CB 3C
App:INFO : Change Key for card key 2 is Successful to Se050ObjID= 2103308288

App:INFO :Reverting Successful
Tx> 0B 00
Tx> 5A 11 22 33
Rx< 0B 00 00
App:INFO :Select the AFC Application Successful

App:INFO :attempting to authenticate with cardkey = 0 and Se0Obj ID =
↪2103308288
Tx> 0A 00
Tx> 71 00 06 00 00 00 00 00 00
Rx< 0A 00 AF A0 64 12 F1 64 01 26 09 9B 74 F6 84 4E AA DA B1
App:INFO :
CARD ==> SE050 16-byte Ek(RndB) =
(Len=16)
A0 64 12 F1 64 01 26 09 9B 74 F6 84 4E AA DA B1
App:INFO :
CARD <===== SE050 E(Kx, RandA || RandB') =
(Len=32)
B7 19 71 EE F0 43 C8 CF 93 C2 9A 9A E2 22 E3 6C
00 93 7E 42 C0 25 16 55 86 78 30 A5 83 DE EB 88
Tx> 0B 00
Tx> AF B7 19 71 EE F0 43 C8 CF 93 C2 9A 9A E2 22 E3 6C 00 93 7E 42 C0 25 16 55 86 78
↪30 A5 83 DE EB 88
Rx< 0B 00 00 06 EF 22 A6 93 17 9D 14 50 C0 B9 34 E7 EA B5 16 3F 2E 8C FE CB 4D 49 1A
↪C5 93 A6 42 6A 01 14 16
App:INFO :
CARD ==> SE050 32-byte E(Kx, TI||RandA'||PDCap2||PCDcap2) =
(Len=32)
06 EF 22 A6 93 17 9D 14 50 C0 B9 34 E7 EA B5 16
3F 2E 8C FE CB 4D 49 1A C5 93 A6 42 6A 01 14 16
App:INFO :
CARD <===== SE050 E(Kx, RandA || RandB') =
(Len=12)
00 00 00 00 00 00 00 00 00 00 00 00
App:INFO :Dumped Session Key is (Len=16)
07 25 BD 23 62 1F B6 B4 AC 19 EF AC 19 BD DA C1
App:INFO :Dumped Session Mac is (Len=16)
3B 7F 15 CD 91 9C 11 A6 79 E0 34 FC 78 E1 62 89
App:INFO :Dumped TI is (Len=4)
03 55 8B AB
App:INFO :pDataParams->wCmdCtr=0
App:INFO : EV2 First Authenticate Successful

Tx> 0A 00
Tx> 77 00

```

(continues on next page)

(continued from previous page)

```

Rx< 0A 00 AF F8 AF 69 A2 31 62 00 75 FB 3F 80 DF AE 3C 2E 4C
 App:INFO :
CARD ==> SE050 16-byte Ek(RndB) =
(Len=16)
 F8 AF 69 A2 31 62 00 75 FB 3F 80 DF AE 3C 2E 4C
 App:INFO :
CARD <===== SE050 E(Kx, RandA || RandB') =
(Len=32)
 CB 08 4D F4 AA D1 42 AB 89 12 22 CA C4 50 26 39
 C5 AA 0C 0B 22 46 34 02 BE 41 26 07 A1 26 3A AB
Tx> 0B 00
Tx> AF CB 08 4D F4 AA D1 42 AB 89 12 22 CA C4 50 26 39 C5 AA 0C 0B 22 46 34 02 BE 41
↪26 07 A1 26 3A AB
Rx< 0B 00 00 23 D9 43 F1 D1 3D 8C B0 24 C4 1C EC E1 5F B4 CC
 App:INFO :
CARD ==> SE050 32-byte E(Kx, TI||RandA'||PDCap2||PCDcap2) =
(Len=16)
 23 D9 43 F1 D1 3D 8C B0 24 C4 1C EC E1 5F B4 CC
 App:INFO :Dumped Session Key is (Len=16)
 3C DB CC D7 9F A1 EB BD A7 79 A7 5C 20 53 B8 E0
 App:INFO :Dumped Session Mac is (Len=16)
 98 CF 64 9A 98 09 5B D4 E3 1E 1B D8 D6 E3 A9 34
 App:INFO :Dumped TI is (Len=4)
 03 55 8B AB
 App:INFO :pDataParams->wCmdCtr=0
 App:INFO : EV2 Following Authenticate Successful

 App:INFO :Authenticated with cardkey = 0 and Se0Obj ID = 2103308288
 App:INFO :attempting to change cardkey = 0 from Old Se050ObjID= 2103308288 to
↪new Se050ObjID= 2103308289
Tx> 0A 00
Tx> C6 00 00 C0 A4 C0 2A 1C 5E 6C 81 EB 86 97 E6 ED EA A4 56 00 30 3F 72 3E 87 7F 1B
↪E7 05 8E 5C 83 A3 42 8F B0 2A 05 5C C2 1F F5 95
Rx< 0A 00 00
 App:INFO : Change Key for card key 0 is Successful to Se050ObjID= 2103308289

 App:INFO :The previous auth session is not valid anymore.
 App:INFO :So.....
 App:INFO :Auth with the changed cardkey 0
Tx> 0B 00
Tx> 5A 11 22 33
Rx< 0B 00 00
 App:INFO :Select the AFC Application Successful

 App:INFO :attempting to authenticate with cardkey = 0 and Se0Obj ID =
↪2103308289
Tx> 0A 00
Tx> 71 00 06 00 00 00 00 00
Rx< 0A 00 AF 2C 85 14 8C FB B8 77 28 04 40 21 3B FA F3 4A A9
 App:INFO :
CARD ==> SE050 16-byte Ek(RndB) =
(Len=16)
 2C 85 14 8C FB B8 77 28 04 40 21 3B FA F3 4A A9
 App:INFO :
CARD <===== SE050 E(Kx, RandA || RandB') =
(Len=32)
 96 70 F8 24 67 30 ED 30 98 AF 2A EC 3F A1 23 32

```

(continues on next page)

(continued from previous page)

```

 37 B4 03 7D E9 A3 8D 3E E2 46 91 04 26 CD A0 26
Tx> 0B 00
Tx> AF 96 70 F8 24 67 30 ED 30 98 AF 2A EC 3F A1 23 32 37 B4 03 7D E9 A3 8D 3E E2 46
↪91 04 26 CD A0 26
Rx< 0B 00 00 6B 86 58 E8 EA 62 B8 FC 13 7F 05 CC 57 38 5B 4E 2F D0 A7 DF 45 02 29 96
↪42 2C 07 D3 E1 F6 B8 3D
 App:INFO :
 CARD =====> SE050 32-byte E(Kx, TI||RndA'||PDCap2||PCDcap2) =
(Len=32)
 6B 86 58 E8 EA 62 B8 FC 13 7F 05 CC 57 38 5B 4E
 2F D0 A7 DF 45 02 29 96 42 2C 07 D3 E1 F6 B8 3D
 App:INFO :
 CARD <===== SE050 E(Kx, RandA || RandB') =
(Len=12)
 00 00 00 00 00 00 00 00 00 00 00 00
 App:INFO :Dumped Session Key is (Len=16)
 C7 98 D4 C6 89 4A CB 9C 7C 7E 4C 2C 3B 19 D5 D1
 App:INFO :Dumped Session Mac is (Len=16)
 7D 9A F9 AA DB A2 BC 68 4F 98 A6 92 7D 92 C4 0C
 App:INFO :Dumped TI is (Len=4)
 17 33 24 D0
 App:INFO :pDataParams->wCmdCtr=0
 App:INFO : EV2 First Authenticate Successful

Tx> 0A 00
Tx> 77 00
Rx< 0A 00 AF CF C0 D0 8F CE C0 26 BE 40 31 0B 0E 3F C2 18 B6
 App:INFO :
 CARD =====> SE050 16-byte Ek(RndB) =
(Len=16)
 CF C0 D0 8F CE C0 26 BE 40 31 0B 0E 3F C2 18 B6
 App:INFO :
 CARD <===== SE050 E(Kx, RandA || RandB') =
(Len=32)
 D5 47 2E 21 4A CD 7F 95 A9 0E 82 E1 34 AB 30 3E
 96 B6 4D 0A 32 B2 9C 7D DB 83 07 7D 74 E8 11 20
Tx> 0B 00
Tx> AF D5 47 2E 21 4A CD 7F 95 A9 0E 82 E1 34 AB 30 3E 96 B6 4D 0A 32 B2 9C 7D DB 83
↪07 7D 74 E8 11 20
Rx< 0B 00 00 49 CF 34 0C 90 9D 1A 8D FF 2C 73 48 2A 41 88 19
 App:INFO :
 CARD =====> SE050 32-byte E(Kx, TI||RndA'||PDCap2||PCDcap2) =
(Len=16)
 49 CF 34 0C 90 9D 1A 8D FF 2C 73 48 2A 41 88 19
 App:INFO :Dumped Session Key is (Len=16)
 34 40 1C 99 E3 5A 4E CD 1A CF 7A 92 4D 73 F6 28
 App:INFO :Dumped Session Mac is (Len=16)
 D3 48 4D 4B 17 4E C0 20 D3 6A 24 7C FC E3 42 DF
 App:INFO :Dumped TI is (Len=4)
 17 33 24 D0
 App:INFO :pDataParams->wCmdCtr=0
 App:INFO : EV2 Following Authenticate Successful

 App:INFO :Authenticated with cardkey = 0 and Se00bj ID = 2103308289
 App:INFO :Checking that the auth session with changed key is valid by trying
↪an encrypted communication
Tx> 0A 00

```

(continues on next page)

(continued from previous page)

```

Tx> 51 D6 09 AF C6 31 06 8A 65
Rx< 0A 00 00 98 2B 05 82 82 E6 B4 BB E2 84 13 93 84 EF A7 68 F3 AA D6 08 8E E4 EA 93
 App:INFO :CARD UID is as below (Len=7)
 04 40 40 42 4D 4D 80
 App:INFO :Encrypted communication with changed key successful
 App:INFO :Reverting the changed cardkey 0
Tx> 0B 00
Tx> 5A 11 22 33
Rx< 0B 00 00
 App:INFO :Select the AFC Application Successful

 App:INFO :attempting to authenticate with cardkey = 0 and Se0Obj ID =
↳2103308289
Tx> 0A 00
Tx> 71 00 06 00 00 00 00 00
Rx< 0A 00 AF 1B 65 22 92 28 22 2E 39 FB BB AF 5F 92 02 0B 48
 App:INFO :
 CARD <=====> SE050 16-byte Ek(RndB) =
 (Len=16)
 1B 65 22 92 28 22 2E 39 FB BB AF 5F 92 02 0B 48
 App:INFO :
 CARD <=====> SE050 E(Kx, RandA || RandB') =
 (Len=32)
 05 FF AE E6 71 6B 12 CF 7A FF B1 AB 7A B0 CF F0
 14 0D A2 CD 70 74 04 09 B8 EB EF 7F 3C 3E 5E FC
Tx> 0B 00
Tx> AF 05 FF AE E6 71 6B 12 CF 7A FF B1 AB 7A B0 CF F0 14 0D A2 CD 70 74 04 09 B8 EB
↳EF 7F 3C 3E 5E FC
Rx< 0B 00 00 BE 19 7B F2 D4 16 7B 3F 6A D3 9B 8A 6A E4 61 D5 BE BE 49 8B 52 56 CA ED
↳FA 4C 43 4D CE 68 ED 62
 App:INFO :
 CARD <=====> SE050 32-byte E(Kx, TI||RandA'||PDCap2||PCDcap2) =
 (Len=32)
 BE 19 7B F2 D4 16 7B 3F 6A D3 9B 8A 6A E4 61 D5
 BE BE 49 8B 52 56 CA ED FA 4C 43 4D CE 68 ED 62
 App:INFO :
 CARD <=====> SE050 E(Kx, RandA || RandB') =
 (Len=12)
 00 00 00 00 00 00 00 00 00 00 00 00
 App:INFO :Dumped Session Key is (Len=16)
 B9 A0 DF 3F F5 AF 09 AB DB E5 F2 8C 95 D6 B3 02
 App:INFO :Dumped Session Mac is (Len=16)
 BC BE C7 49 26 D2 C2 0D 12 2B A9 D9 8C 86 44 76
 App:INFO :Dumped TI is (Len=4)
 3D 06 B4 0F
 App:INFO :pDataParams->wCmdCtr=0
 App:INFO : EV2 First Authenticate Successful

Tx> 0A 00
Tx> 77 00
Rx< 0A 00 AF A9 AA 26 E3 BE 9E 1F 08 05 18 F2 18 BB BC C8 8C
 App:INFO :
 CARD <=====> SE050 16-byte Ek(RndB) =
 (Len=16)
 A9 AA 26 E3 BE 9E 1F 08 05 18 F2 18 BB BC C8 8C
 App:INFO :
 CARD <=====> SE050 E(Kx, RandA || RandB') =

```

(continues on next page)

(continued from previous page)

```

(Len=32)
 C6 7D DD FE 93 2C 1B 1D AE 73 E9 55 57 21 5B 67
 EB 77 CC CB 88 BD FD C2 D1 C6 63 79 8F 56 D6 C1
Tx> 0B 00
Tx> AF C6 7D DD FE 93 2C 1B 1D AE 73 E9 55 57 21 5B 67 EB 77 CC CB 88 BD FD C2 D1 C6
↪63 79 8F 56 D6 C1
Rx< 0B 00 00 50 1F 48 BB 26 A9 33 96 F8 F0 F6 34 63 83 FE 0A
 App:INFO :
 CARD =====> SE050 32-byte E(Kx, TI||RndA'||PDCap2||PCDcap2) =
(Len=16)
 50 1F 48 BB 26 A9 33 96 F8 F0 F6 34 63 83 FE 0A
 App:INFO :Dumped Session Key is (Len=16)
 08 C2 12 46 4F 51 3C A9 15 36 9F 4A 61 97 BA C2
 App:INFO :Dumped Session Mac is (Len=16)
 91 1E E8 D8 88 58 E4 0A BB F6 93 B5 FA B5 C9 0B
 App:INFO :Dumped TI is (Len=4)
 3D 06 B4 0F
 App:INFO :pDataParams->wCmdCtr=0
 App:INFO : EV2 Following Authenticate Successful

 App:INFO :Authenticated with cardkey = 0 and Se0Obj ID = 2103308289
 App:INFO :attempting to change cardkey = 0 from Old Se050ObjID= 2103308289 to
↪new Se050ObjID= 2103308288
Tx> 0A 00
Tx> C6 00 00 42 A7 99 85 84 E2 CA 46 3F D0 E5 12 59 62 F3 17 5E D6 C4 16 E5 09 49 8B
↪0D AD AD 73 75 20 A1 E7 43 06 EC 4D BC 1A 07 D1
Rx< 0A 00 00
 App:INFO : Change Key for card key 0 is Successful to Se050ObjID= 2103308288

 App:INFO :Reverting Successful
 App:INFO : Auth session is reset in software
 App:INFO : Auth session is killed in SE
 App:INFO :ex_sss Finished

```

## 5.36 Tool to create Reference key file

This tool is to demonstrate how to implement a command-line utility for native systems. This utility can be used to generate/inject keypair and create reference key files.

This is beneficial for environments which do not have python installed.

---

**Note:** This example is implemented only for NIST-P256 curve. It can only be compiled when Host Crypto is OpenSSL.

---

### 5.36.1 Building the example

Use the following CMake configurations to compile the example

- CMake configurations: `WithHostCrypto_OPENSSL: ON`
- Project: `seTool`

### 5.36.2 How to use

This example provides four command-line parameters to select the operation to perform.

- 1) To **generate** a keypair, run the tool as:

```
seTool genECC <keyId>
```

Where:

- *keyId* is the keypair index at which we want to generate the keypair.

- 2) To **inject** a keypair, run the tool as:

```
seTool setECC <keyId> <filename>
```

Where:

- *keyId* is the keypair index at which we want to inject the keypair
- *filename* is the path of the file in which keypair is stored in PEM format.

- 3) To **retrieve the public key**, run the tool as:

```
seTool getPublic <keyId> <filename>
```

Where:

- *keyId* is the keypair index from which we want to retrieve the public key
- *filename* is the path of the file in which we want to store the key in PEM format.

- 4) To **create a reference key** for an injected keypair, run the tool as:

```
seTool getRef <keyId> <filename>
```

Where:

- *keyId* is the keypair index at which we keypair is stored and
- *filename* is the path of the file in which we want to store the reference key in PEM format.

The generated reference key can be used by OpenSSL Engine.

## 5.37 Building a self-signed certificate

This demo is to demonstrate how we can use provisioned keys to create a self-signed certificate to communicate with cloud platforms. In this example, we use two binaries, one to generate a keypair inside the secure element and another to use the generated keypair to create a self-signed certificate.

---

**Note:** We use OpenSSL in this example to create the certificate.

---

### 5.37.1 How to use

1. Run the binary `generate_certificate_key` to generate an ECC-256 keypair inside the secure element.
2. Run the binary `generate_certificate` to create a self-signed certificate. This demo provisions the the generated certificate into the secure element.
3. You can read-out the certificate using SSS-APIs from keyId `CERTIFICATE_KEY_ID + 1` as defined in `certificate.h` file.

## 5.38 Write APDU to buffer

Normally, when we create an APDU, we also perform transceive operation and return the result to the application. Optionally, the user may want to just create the APDU and not perform the transceive operation.

This can be done by creating your own transaction function. All SE05x APIs call a common transaction function in which the complete APDU is created and then transmitted.

The following function can be used as reference.

```
smStatus_t fLogTransmitBuffer(Se05xSession_t *pwrite_apdubufferctx,
 const tlvHeader_t *hdr,
 uint8_t *cmdBuf,
 size_t cmdBufLen,
 uint8_t *rsp,
 size_t *rspLen,
 uint8_t hasle)
{
 memset(gTxBuffer, 0, sizeof(gTxBuffer));
 size_t i = 0;
 memcpy(&gTxBuffer[i], hdr, sizeof(*hdr));
 smStatus_t ret = SM_OK;
 i += sizeof(*hdr);
 if (cmdBufLen > 0) {
 // The Lc field must be extended in case the length does not fit
 // into a single byte (Note, while the standard would allow to
 // encode 0x100 as 0x00 in the Lc field, nobody who is sane in his mind
 // would actually do that).
 if ((cmdBufLen < 0xFF) && !hasle) {
 gTxBuffer[i++] = (uint8_t)cmdBufLen;
 }
 else {
 gTxBuffer[i++] = 0x00;
 gTxBuffer[i++] = 0xFFu & (cmdBufLen >> 8);
 gTxBuffer[i++] = 0xFFu & (cmdBufLen);
 }
 }
}
```

(continues on next page)

(continued from previous page)

```

 }
 memcpy(&gTxBuffer[i], cmdBuf, cmdBufLen);
 i += cmdBufLen;
}
if (hasle) {
 gTxBuffer[i++] = 0x00;
 gTxBuffer[i++] = 0x00;
}
ret = SM_OK;
gTxBufferLen = i;

return ret;
}

```

Assign this function to session context and use as following

```

Se05xSession_t write_apdubufferctx = {0};
write_apdubufferctx.fp_TXn = &fLogTransmitBuffer;

Se05x_API_WriteBinary(&write_apdubufferctx, &policy, objectID, offset, length,
↳ inputData, inputDataLen);

```

After this, whichever SE05x API you call with write\_apdubufferctx, it will just create the APDU and write it to the buffer. No transceive operation will be performed.

### 5.38.1 Building

Build the project with the following configurations.

#### se05x\_GetAPDUBuffer

- SCP = None
- Project = se05x\_GetAPDUBuffer

### 5.38.2 Running

On successful execution you can see the APDU buffer printed out:

```

App :INFO :Policy for ReadOnly File:
App :INFO :Locked = READ
App :INFO :Excluded = DELETE, WRITE.
App :INFO :pPolicyBuffer (Len=9)
08 00 00 00 00 00 20 00 00
App :INFO :gTxBuffer (Len=38)
80 01 06 00 21 11 09 08 00 00 00 00 00 20 00 00
41 04 11 22 33 44 43 02 00 0A 44 0A 01 02 03 04
05 06 07 08 09 0A

```



## 5.39 Ease of Use configuration - IBM Watson

### 5.39.1 Configuring Device type and Device name

Follow steps given in [How to get SE Platform Information and UID](#) to get the device unique ID.

The device name to be registered on the cloud is the 18-byte UID output in uppercase. In this case, 04005001F5EA9CC8CA7B33042C0559550000.

Device type would change based on which type of keys are being used (EC/RSA2K/RSA4K). Device type is set as NXP-SE050-<type>-D, where type is EC, RSA2K or RSA4K based on the type of key. Refer to section [Trust provisioned KeyIDs](#) for keyIDs of trust provisioned keys and certificates. On your IoT platform, create a new device type and create a new device under it with names as obtained in the previous step.

---

**Note:** If you wish to create a gateway on IoT platform, change Device type as NXP-SE050-<type>-G, where type is EC, RSA2K or RSA4K based on the type of key.

---

### 5.39.2 Uploading certificate chain

Certificate chain for cloud connection can be found under demos/Certificate\_Chains/0004\_A1F4 directory.

Also see [Certificate Chains : DEV Kit](#) for details about certificate chain.

---

**Note:** A1F4 is the OEF number. The directory name may change based on your OEF configuration.

---

The certificate chain for ECC from RootCA to Device certificate is as:

```
IOT_NXP-01-CERT_IOT_CA_KEY-IoTRootCAvE305-01-20190320162439-EC_SEC_P384R1-4B7E5A.
crt -> IOT_NXP-01-CERT_IOT_4LAYER_CA_KEY-IoTInt4LAYERCAvE205-01-20190320164314-EC_SEC_P256R1
crt -> CloudConn-Intermediate-ECC_OEF_A1F4.crt -> ECC Device/Gateway Certificate
```

The certificate chain for RSA from RootCA to Device certificate is as:

```
IOT_NXP-01-CERT_IOT_CA_KEY-IoTRootCAvR406-01-20190425163255-RSA4096-BAB872.crt
-> IOT_NXP-01-CERT_IOT_4LAYER_CA_KEY-IoTInt4LayerCAvR406-01-20190425163534-RSA4096-540F19.
crt -> CloudConn-Intermediate-RSA_OEF_A1F4.crt -> RSA Device/Gateway Certificate
```

In your IoT platform, go to settings -> CA Certificates section and upload the certificate chain (RootCA and Intermediate CA certificates) for the device certificate.

### 5.39.3 Running the Demo

This step is only for Linux platforms. If you wish to use an embedded microcontroller, continue to the next step

- Create a reference key file to be used with OpenSSL engine:

```
sscli connect se050 tloi2c none
sscli refpem ecc/rsa pair <trust_provisioned_keyid> keyref.pem
sscli disconnect
```

- Build the OpenSSL engine:

```
cd simw-top
python scripts/create_cmake_projects.py
cd ../simw-top_build/<board>_native_se050_t1oi2c
cmake --build .
make install
ldconfig /usr/local/lib
```

- Based on OpenSSL version, select the appropriate configuration file in <MW\_SRC\_DIR>/simw-top/demos/linux/common directory:

```
openssl11_sss_se050.cnf ----- OpenSSL 1.1.1 and SE050
openssl_sss_se050.cnf ----- OpenSSL 1.0.0 and SE050
```

- Set the openssl config path as:

```
$ export OPENSSL_CONF=/simw-top/demos/linux/common/<appropriate-cnf-file>
```

- To run the demo, see *Running the Demo on iMX/Raspberry Pi*

### 5.39.4 Update cloud example

In file `demos/ksdk/ibm_watson/ibm_watson_iot_config.h`, update the **broker endpoint** and **client ID** according to your account, and **keyIDs** of Trust provisioned keys and certificates used (as obtained from *Trust provisioned KeyIDs*):

```
#define WATSONIOT_MQTT_BROKER_ENDPOINT "leohx6.messaging.internetofthings.ibmcloud.com
↪"
```

```
#define WatstonechoCLIENT_ID \
 "d:leohx6:NXP-SE050-EC-D:377813914287991534125055" ///< MQTT client ID should be
↪unique for every device
```

```
#define SSS_KEYPAIR_INDEX_CLIENT_PRIVATE 0x20181003 //keyID of device keypair
#define SSS_CERTIFICATE_INDEX 0x20181004 //keyID of device certificate
```

### 5.39.5 Build and run the demo.

Build and run `demo_ibm_watson`.

CMake configurations:

- RTOS\_FreeRTOS: ON
- WithHostCrypto\_MBEDTLS: ON
- Withmbedtls\_ALT\_SSS: ON
- IOT\_IBM: ON

## 5.40 Ease of Use configuration - Google Cloud Platform

### 5.40.1 Pre-requisites

- Google Cloud Platform Account
- pySSSCLI Tool. Refer to *CLI Tool*

### 5.40.2 Creating Registry and Devices

- If you are using an embedded microcontroller, flash VCOM binary present in `binaries` folder onto the board.
- Read out the device certificate from the SE using pySSSCLI Tool
  - Refer to section *Trust provisioned KeyIDs* for keyIDs of trust provisioned certificates.
  - Read out the trust provisioned certificate as:

```
ssscli connect se050 <conn-type> COMxx
ssscli get cert <trust_provisioned_keyid> <filename>
ssscli disconnect
```

---

**Note:** Give connection parameters according to your board. Refer to *List of ssscli commands* for details on supported parameters.

---

- Device certificate will be stored at the file location provided.

---

**Note:** Give extension of the filename as `.cer` to store in *PEM* format

---

- Create a registry on Google Cloud Platform and upload the intermediate certificate:
  - Intermediate certificates are located in `demos/Certificate_Chains/0004_A1F4` directory. Based on device certificate, ECC or RSA, the intermediate certificates are `CloudConn-Intermediate-ECC_OEF_A1F4.crt` or `CloudConn-Intermediate-RSA_OEF_A1F4.crt` respectively.
  - Create a device in that registry and upload the device certificate obtained in the second step.

Also see *Certificate Chains : DEV Kit* for details about certificate chain.

### 5.40.3 Running the Demo

This step is only for Linux platforms. If you wish to use an embedded microcontroller, continue to the next step

- Create a reference key file to be used with OpenSSL engine:

```
ssscli connect se050 tloi2c none
ssscli refpem ecc/rsa pair <trust_provisioned_keyid> keyref.pem
ssscli disconnect
```

- Build the OpenSSL engine:

```
cd simw-top
python scripts/create_cmake_projects.py
cd ../simw-top_build/<board>_native_se050_t1oi2c
cmake --build .
make install
ldconfig /usr/local/lib
```

- Based on OpenSSL version, select the appropriate configuration file in <MW\_SRC\_DIR>/simw-top/demos/linux/common directory:

```
openssl11_sss_se050.cnf ----- OpenSSL 1.1.1 and SE050
openssl_sss_se050.cnf ----- OpenSSL 1.0.0 and SE050
```

- Set the openssl config path as:

```
$ export OPENSSL_CONF=/simw-top/demos/linux/common/<appropriate-cnf-file>
```

- To run the demo, see *Building the application*

#### 5.40.4 Update cloud example

In file demos/ksdk/gcp/gcp\_iot\_config.h, update the **project name**, **location name**, **registry name** and **device name** according to your account, and **keyIDs** of Trust provisioned keys and certificates used (as obtained from *Trust provisioned KeyIDs*):

```
#define GCP_PROJECT_NAME "pgh-cloud-iot"
#define GCP_LOCATION_NAME "us-central1"
#define GCP_REGISTRY_NAME "nxp-se-demo-reg"

#if (SSS_HAVE_SE050_C || SSS_HAVE_SE050_A)
#define GCP_DEVICE_NAME "nxp-ecc-dev-01"
#elif SSS_HAVE_SE050_B
#define GCP_DEVICE_NAME "nxp-rsa-dev-01"
#else
#define GCP_DEVICE_NAME "a71ch-dev-04"
#endif
```

```
#define SSS_KEYPAIR_INDEX_CLIENT_PRIVATE 0x20181001
#define SSS_CERTIFICATE_INDEX 0x20181002
```

#### 5.40.5 Build and run the demo.

Build and run demo\_gcp.

CMake configurations:

- RTOS\_FreeRTOS: ON
- WithHostCrypto\_MBEDTLS: ON
- Withmbedtls\_ALT\_SSS: ON
- IOT\_GCP: ON

## 5.41 Ease of Use configuration - Azure IoT Hub

### 5.41.1 Creating Device on Azure DPS

Follow *Raspberry Pi Build* or *i.MX Linux Build* to prepare your board.

You will need to read-out the device certificates using pyCLI tool.

If you wish to use an embedded microcontroller, follow the steps given below.

- Flash vcom binary present in `binaries` folder onto the board.
- Read out the device certificate from the SE using `ssscli.exe` present in `binaries/pySSSCLI` directory.

Refer to section *Trust provisioned KeyIDs* for keyIDs of trust provisioned certificates.

- Read out the trust provisioned certificate as:

```
ssscli connect se050 vcom COMxx
ssscli get cert <trust_provisioned_keyid> <filename>
ssscli disconnect
```

---

**Note:** Give connection parameters according to your board. Refer to *List of ssscli commands* for details on supported parameters.

---

- Device certificate will be stored at the file location provided.

---

**Note:** Give extension of the filename as `.cer` to store in *PEM* format

---

- Parse the extracted certificate using a cryptography tool such as OpenSSL to see the subject common name. This common name should be the name of the device registered on the Azure DPS.
- On your Azure portal, go to Azure DPS and create an *Individual Enrollment*. Enter the **DeviceID** as the subject name extracted in the previous step and in **Primary Certificate**, upload the extracted device certificate. You do not need to upload secondary certificate.

---

**Note:** Ensure that your DPS is linked to IoT Hub.

---

Select the device as an **Edge device** and save the configuration.

### 5.41.2 Registering Device to IoT Hub

- To register the saved device to IoT Hub, you would need a Linux platform. Run the following command to create a reference key file to be used with OpenSSL engine:

```
ssscli connect se050 tloi2c none
ssscli refpem ecc/rsa pair <trust_provisioned_keyid> keyref.pem
ssscli disconnect
```

- Build the OpenSSL engine:

```
cd simw-top
python scripts/create_cmake_projects.py
cd ../simw-top_build/<board>_native_se050_t1oi2c
cmake --build .
make install
ldconfig /usr/local/lib
```

- Based on OpenSSL version, select the appropriate configuration file in <MW\_SRC\_DIR>/simw-top/demos/linux/common directory:

```
openssl11_sss_se050.cnf ----- OpenSSL 1.1.1 and SE050
openssl_sss_se050.cnf ----- OpenSSL 1.0.0 and SE050
```

- Set the openssl config path as:

```
$ export OPENSSL_CONF=/simw-top/demos/linux/common/<appropriate-cnf-file>
```

Follow the steps listed in *Create device enrollment in azure IoT Hub portal* to register your device to the linked IoT Hub.

If you do not have a linux platform, follow the steps listed in <https://docs.microsoft.com/en-us/azure/iot-dps/tutorial-provision-device-to-hub> to register your device to IoT Hub.

### 5.41.3 Running Azure Demo

**This step is only for Linux platforms. If you wish to use an embedded microcontroller, continue to the next step**

Running Azure registration application successfully would have generated a JSON file to connect to your device. Use this JSON file to connect to Azure as:

```
./azure_imx_connect --json <JSON-file>
```

Where **JSON-file** is the generated file.

### 5.41.4 Update cloud example

In file `demos/ksdk/azure/azure_iot_config.h`, update `AZURE_IOT_HUB_NAME`, `AZURE_LOCATION_NAME` and `AZURE_DEVICE_NAME` according to your credentials and update **keyIDs** of Trust Provisioned keys and certificates used (as obtained from *Trust provisioned KeyIDs*):

```
#define AZURE_IOT_HUB_NAME "SIMW-JENKINS-HUB"
#define AZURE_LOCATION_NAME "bengaluru"
#define AZURE_DEVICE_NAME "70850359433640212281259"
```

```
#define AZURE_IOT_KEY_INDEX_SM 0x223344 ///< Index where client key is kept
↪ //Decimal - 2241348
#define AZURE_IOT_CLIENT_CERT_INDEX_SM 0x223345 ///< Index where client certificate
↪ is kept //Decimal - 2241349
```

### 5.41.5 Build and run the demo.

Build and run `azure_demo`.

CMake configurations:

- `RTOS_FreeRTOS: ON`
- `WithHostCrypto_MBEDTLS: ON`
- `Withmbedtls_ALT_SSS: ON`
- `IOT_AZURE: ON`

## 5.42 Ease of Use configuration - AWS IoT Console

### 5.42.1 Pre-requisites

- AWS IoT Console Account
- Active Multi-Account Registration feature for your account. For more information, refer to [Multi-Account Registration](#)
- AWS CLI Setup on your machine
- pySSSCLI Tool

### 5.42.2 Extracting Device Certificate

Using pySSSCLI Tool, read out the device certificate. Refer to *Trust provisioned KeyIDs* for keyIDs of trust provisioned keys and certificates.

---

**Note:** If you wish to use an embedded microcontroller, flash the VCOM binary on your board first. VCOM binaries are available in `binaries` directory.

---

Extract the device certificate as:

```
ssscli connect se05x <conn-type> <port>
ssscli get cert <certificate-keyId> <certificate-filename>
ssscli disconnect
```

---

**Note:** Give connection parameters according to your board. Refer to *List of ssscli commands* for details on supported parameters.

---

### 5.42.3 Registering Device Certificate

Use AWS CLI Tool to register the extracted device certificate on to your AWS IoT Console:

```
aws iot register-certificate-without-ca --certificate-pem <certificate-filename> --
↪status ACTIVE
aws iot attach-policy --target <certificate ARN> --policy-name <policy name>
```

---

**Note:** Certificate ARN will be printed out after execution of the first command

---

Run the following command to print out the SNI string. This will be used later:

```
aws iot describe-endpoint --endpoint-type iot:Data-ATS
```

### 5.42.4 Running on Linux

**This step is only for Linux platforms. If you wish to use an embedded microcontroller, continue to the next step**

- Run the following commands to build OpenSSL engine for SE050:

```
cd simw-top
python scripts/create_cmake_projects.py
cd ../simw-top_build/<board>_native_se050_t1oi2c
cmake --build .
make install
ldconfig /usr/local/lib
```

- Navigate to `demos/linux/aws_eou` directory and execute the `buildScript.sh` as:

```
./buildScript.sh
```

This will build the project `iot_demo_mqtt`.

- Based on OpenSSL version, select the appropriate configuration file in `<MW_SRC_DIR>/simw-top/demos/linux/common` directory:

```
openssl11_sss_se050.cnf ----- OpenSSL 1.1.1 and SE050
openssl_sss_se050.cnf ----- OpenSSL 1.0.0 and SE050
```

- Set the openssl config path as:

```
$ export OPENSSL_CONF=/simw-top/demos/linux/common/<appropriate-cnf-file>
```

- Create a reference file of device keypair for OpenSSL engine:

```
ssscli connect se05x <conn-type> <port>
ssscli refpem ecc/rsa pair <keypair-keyId> <ref-filename>
ssscli disconnect
```

---

**Note:** Make sure that the device keypair used corresponds to the device certificate

---

- Run the demo:



```
cd output/bin
./iot_demo_mqtt -i "ThingName" -h <endpoint> -r AmazonRootCA1.pem -c <certificate-
↪filename> -k <ref-filename>
```

where `endpoint` is the SNI string obtained in step [Registering Device Certificate](#), `certificate-filename` is the device certificate extracted in step [Extracting Device Certificate](#) and `ref-filename` is the reference key created in the previous step.

### 5.42.5 Update Cloud Example

- Update the `clientcredentialMQTT_BROKER_ENDPOINT` variable in `demos/ksdk/common/aws_clientcredential.h` file with the SNI string obtained in previous step.
- Update `SSS_KEYPAIR_INDEX_CLIENT_PRIVATE` and `SSS_CERTIFICATE_INDEX_CLIENT` in file `demos/ksdk/common/aws_iot_config.h` with KeyIDs of device keypair and device certificate respectively.

```
#define SSS_KEYPAIR_INDEX_CLIENT_PRIVATE 0x20181005
#define SSS_CERTIFICATE_INDEX_CLIENT 0x20181007
```

### 5.42.6 Build and run the demo.

Build and run project `aws_demo`.

CMake configurations:

- `RTOS_FreeRTOS: ON`
- `WithHostCrypto_MBEDTLS: ON`
- `Withmbedtls_ALT_SSS: ON`
- `IOT_AWS: ON`



## PLUGINS / ADD-INS

Plugins / Add-ins for other platforms

### 6.1 Introduction on OpenSSL engine

Starting with OpenSSL 0.9.6 an ‘Engine interface’ was added to OpenSSL allowing support for alternative cryptographic implementations. This Engine interface can be used to interface with external crypto devices. The key injection process is secure module specific and is not covered by the Engine interface.

Depending on the capabilities of the attached secure element (e.g. SE050\_C, A71CH, ...) the following functionality can be made available over the OpenSSL Engine interface:

- EC crypto
  - EC sign/verify
  - ECDH compute key
- RSA crypto
  - RSA sign/verify
  - RSA priv\_key\_decrypt/pub\_key\_encrypt
- Fetching random data

#### 6.1.1 General

##### OpenSSL versions

The OpenSSL Engine is compatible with OpenSSL versions 1.0.2 or 1.1.1.

##### OpenSSL Configuration file

It’s possible to add OpenSSL engine specific extensions to the OpenSSL configuration file. Using these extensions one can control whether the supported crypto functionality is delegated to the Secure Element or whether it is handled by the OpenSSL SW implementation.

The actual contents of the configuration file depends on the OpenSSL version and the attached secure element (SE050 or A71CH). The `demos/linux/common` folder of this SW package contains 4 reference configuration files covering both SE050 and A71CH for the two supported OpenSSL versions.

The following configuration file fragment (extracted from `openssl111_sss_se050.cnf`) highlights the required changes to enable the full functionality of the SE050\_C OpenSSL Engine on an iMX Linux system:

```
...
System default
openssl_conf = nxp_engine
...

...
[nxp_engine]
engines = engine_section

[engine_section]
e4sss_se050 = e4sss_se050_section

[e4sss_se050_section]
engine_id = e4sss
dynamic_path = /usr/local/lib/libsss_engine.so
init = 1
default_algorithms = RAND,RSA,EC
```

One overrules the default OpenSSL configuration file by setting the environment variable `OPENSSL_CONF` to the path of the custom configuration file.

## Platforms

The OpenSSL engine can be used on iMX boards (running Linux) or on Raspberry Pi (running Raspbian).

### 6.1.2 Keys

#### Key Management

The cryptographic functionality offered by the OpenSSL engine requires a reference to a key stored inside the Secure Element (exception is `RAND_Method`). These keys are typically inserted into the Secure Element in a secured environment during production.

OpenSSL requires a key pair, consisting of a private and a public key, to be loaded before the cryptographic operations can be executed. This creates a challenge when OpenSSL is used in combination with a secure element as the private key cannot be extracted out from the Secure Element.

The solution is to populate the OpenSSL Key data structure with only a reference to the Private Key inside the Secure Element instead of the actual Private Key. The public key as read from the Secure Element can still be inserted into the key structure.

OpenSSL crypto API's are then invoked with these data structure objects as parameters. When the crypto API is routed to the Engine, the OpenSSL engine implementation decodes these key references and invokes the SSS API with correct Key references for a cryptographic operation.

## EC Reference key format

The following provides an example of an EC reference key. The value reserved for the private key has been used to contain:

- a pattern of 0x10...00 to fill up the datastructure MSB side to the desired key length
- a 32 bit key identifier (in the example below 0x7DCCBBAA)
- a 64 bit magic number (always 0xA5A6B5B6A5A6B5B6)
- a byte to describe the key class (0x10 for Key pair and 0x20 for Public key)
- a byte to describe the key index (use a reserved value 0x00)

```
Private-Key: (256 bit)
priv:
 10:00:00:00:00:00:00:00:00:00:00:00:00:00:00:
 00:00:00:7D:CC:BB:AA:A5:A6:B5:B6:A5:A6:B5:B6:
 kk:ii
pub:
 04:1C:93:08:8B:26:27:BA:EA:03:D1:BE:DB:1B:DF:
 8E:CC:87:EF:95:D2:9D:FC:FC:3A:82:6F:C6:E1:70:
 A0:50:D4:B7:1F:F2:A3:EC:F8:92:17:41:60:48:74:
 F2:DB:3D:B4:BC:2B:F8:FA:E8:54:72:F6:72:74:8C:
 9E:5F:D3:D6:D4
ASN1 OID: prime256v1
```

### Note:

- The key identifier 0x7DCCBBAA (stored in big-endian convention) is in front of the magic number 0xA5A6B5B6A5A6B5B6
- The padding of the private key value and the magic number make it unlikely a normal private key value matches a reference key.
- Ensure the value reserved for public key and ASN1 OID contain the values matching the stored key.

## RSA Reference key format

The following provides an example of an RSA reference key.

- The value reserved for 'p' (aka 'prime1') is used as a magic number and is set to '1'
- The value reserved for 'q' (aka 'prime2') is used to store the 32 bit key identifier (in the example below 0x6DCCBB11)
- The value reserved for '(inverse of q) mod p' (aka 'IQMP' or 'coefficient') is used to store the magic number 0xA5A6B5B6

```
Private-Key: (2048 bit)
modulus:
 00:b5:48:67:f8:84:ca:51:ac:a0:fb:d8:e0:c9:a7:
 72:2a:bc:cb:bc:93:3a:18:6a:0f:a1:ae:d4:73:e6:
 ...
publicExponent: 65537 (0x10001)
privateExponent:
 58:7a:24:39:90:f4:13:ff:bf:2c:00:11:eb:f5:38:
```

(continues on next page)

(continued from previous page)

```

 b1:77:dd:3a:54:3c:f0:d5:27:35:0b:ab:8d:94:93:
 ...
prime1: 1 (0x1)
prime2: 1842133777 (0x6DCCBB11)
exponent1:
 00:c1:c9:0a:cc:9f:1a:c5:1c:53:e6:c1:3f:ab:09:
 db:fb:20:04:38:2a:26:d5:71:33:cd:17:a0:94:bd:
 ...
exponent2:
 24:95:f0:0b:b0:78:a9:d9:f6:5c:4c:e0:67:d8:89:
 c1:eb:df:43:54:74:a0:1c:43:e3:6f:d5:97:88:55:
 ...
coefficient: 2779166134 (0xA5A6B5B6)

```

---

**Note:**

- Ensure keylength, the value reserved for (private key) modulus and public exponent match the stored key.
  - The mathematical relation between the different key components is not preserved.
  - Setting prime1 to '1' makes it impossible that a normal private key matches a reference key.
- 

## 6.1.3 Building the OpenSSL engine

The cmake build system will create an OpenSSL engine for supported platforms. The resulting OpenSSL engine will be copied to the SW tree in directory `simw-top/sss/plugin/openssl/bin`.

A subsequent `make install` will copy the OpenSSL engine to a standard directory on the file system, in case of iMX Linux e.g. `/usr/local/lib`.

---

**Note:** Ensure the following flag is defined when building an application that will be linked against the engine:  
`-DOPENSSL_LOAD_CONF`

---

## 6.1.4 Sample scripts to demo OpenSSL Engine

The directory `simw-top/sss/plugin/openssl/scripts` contains a set of python scripts. These scripts use the OpenSSL Engine in the context of standard OpenSSL utilities. They illustrate using the OpenSSL Engine for fetching random data, EC or RSA crypto operations. The scripts that illustrate EC or RSA crypto operations depend on prior provisioning of the secure element.

As an example, the following set of commands first creates and provisions EC key material. Then it invokes the OpenSSL Engine for ECDSA sign / verify operations and ECDH calculations. It assumes an SE050 is connected via I2C to an iMX6UL-EVK board:

```

python3 openssl_provisionEC.py --key_type prime256v1
python3 openssl_EccSign.py --key_type prime256v1
python3 openssl_Ecdh.py --key_type prime256v1

```

Further details on using these scripts can be found in the following:

## openssl\_rnd.py

usage: openssl\_rnd.py [-h] [--connection\_data CONNECTION\_DATA]

Generate few random numbers from the attached secure element.

### optional arguments:

**-h, --help** show this help message and exit

**--connection\_data CONNECTION\_DATA** Parameter to connect to SE => eg. COM3, 127.0.0.1:8050, none. Default: none

Example invocation:

```
python openssl_rnd.py --connection_data 127.0.0.1:8050
```

## openssl\_provisionEC.py

usage: openssl\_provisionEC.py [-h] --key\_type KEY\_TYPE [--connection\_type CONNECTION\_TYPE] [--connection\_data CONNECTION\_DATA] [--subsystem SUBSYSTEM]

Provision attached secure element with EC keys

This example generates a complete set of ECC key files (\*.pem) (existing ones overwritten). Performs debug reset the attached secure element. Attached secure element provisioned with EC key. Creates reference key from the injected EC key.

### optional arguments:

**-h, --help** show this help message and exit

### required arguments:

**--key\_type KEY\_TYPE** Supported key types => prime192v1, secp224r1, prime256v1, secp384r1, secp521r1, brainpoolP160r1, brainpoolP192r1, brainpoolP224r1, brainpoolP256r1, brainpoolP320r1, brainpoolP384r1, brainpoolP512r1, secp160k1, secp192k1, secp224k1, secp256k1

### optional arguments:

**--connection\_type CONNECTION\_TYPE** Supported connection types => tloi2c, sci2c, vcom, jrcpv1, jrcpv2, pcsc. Default: tloi2c

**--connection\_data CONNECTION\_DATA** Parameter to connect to SE => eg. COM3, 127.0.0.1:8050, none. Default: none

**--subsystem SUBSYSTEM** Supported subsystem => se050, a71ch, mbedtls. Default: se050

Example invocation:

```
python openssl_provisionEC.py --key_type prime256v1
python openssl_provisionEC.py --key_type prime256v1 --connection_data 169.254.0.1:8050
python openssl_provisionEC.py --key_type secp224k1 --connection_type jrcpv2 --
->connection_data 127.0.0.1:8050
python openssl_provisionEC.py --key_type brainpoolP256r1 --connection_data COM3
python openssl_provisionEC.py --key_type prime256v1 --subsystem a71ch
```

## openssl\_EccSign.py

**usage:** openssl\_EccSign.py [-h] --key\_type KEY\_TYPE [--connection\_data CONNECTION\_DATA]

Validation of Sign Verify with OpenSSL engine using EC Keys

This example showcases sign using reference key, then verify using openssl and vice versa.

### Precondition:

- Inject keys using openssl\_provisionEC.py.

### optional arguments:

**-h, --help** show this help message and exit

### required arguments:

**--key\_type KEY\_TYPE** Supported key types => prime192v1, secp224r1, prime256v1, secp384r1, secp521r1, brainpoolP160r1, brainpoolP192r1, brainpoolP224r1, brainpoolP256r1, brainpoolP320r1, brainpoolP384r1, brainpoolP512r1, secp160k1, secp192k1, secp224k1, secp256k1

### optional arguments:

**--connection\_data CONNECTION\_DATA** Parameter to connect to SE => eg. COM3, 127.0.0.1:8050, none. Default: none

Example invocation:

```
python openssl_EccSign.py --key_type prime256v1
python openssl_EccSign.py --key_type secp160k1 --connection_data 127.0.0.1:8050
```

## openssl\_Ecdh.py

**usage:** openssl\_Ecdh.py [-h] --key\_type KEY\_TYPE [--connection\_data CONNECTION\_DATA]

Validation of ECDH with OpenSSL engine using EC keys

This example showcases ECDH between openssl engine and openssl.

### Precondition:

- Inject keys using openssl\_provisionEC.py.

### optional arguments:

**-h, --help** show this help message and exit

### required arguments:

**--key\_type KEY\_TYPE** Supported key types => prime192v1, secp224r1, prime256v1, secp384r1, secp521r1, brainpoolP160r1, brainpoolP192r1, brainpoolP224r1, brainpoolP256r1, brainpoolP320r1, brainpoolP384r1, brainpoolP512r1, secp160k1, secp192k1, secp224k1, secp256k1

### optional arguments:

**--connection\_data CONNECTION\_DATA** Parameter to connect to SE => eg. COM3, 127.0.0.1:8050, none. Default: none

Example invocation:



```
python openssl_Ecdh.py --key_type prime256v1
python openssl_Ecdh.py --key_type secp160k1 --connection_data 127.0.0.1:8050
```

## **ecc\_all.py**

**usage:** **ecc\_all.py** [-h] [--connection\_type CONNECTION\_TYPE] [--connection\_data CONNECTION\_DATA] [--subsystem SUBSYSTEM]

Validation of OpenSSL Engine using EC keys

This example injects keys with different supported EC Curves, then showcases ECDH & ECDSA using those keys.

### **optional arguments:**

**-h, --help** show this help message and exit

**--connection\_type CONNECTION\_TYPE** Supported connection types => t1oi2c, sci2c, vcom, jrcpv1, jrcpv2, pcsc. Default: t1oi2c

**--connection\_data CONNECTION\_DATA** Parameter to connect to SE => eg. COM3, 127.0.0.1:8050, none. Default: none

**--subsystem SUBSYSTEM** Supported subsystem => se050, a71ch. Default: se050

Example invocation:

```
python ecc_all.py
python ecc_all.py --connection_data 169.254.0.1:8050
python ecc_all.py --connection_data 127.0.0.1:8050 --connection_type jrcpv2
python ecc_all.py --connection_data COM3
```

## **openssl\_provisionRSA.py**

**usage:** **openssl\_provisionRSA.py** [-h] --key\_type KEY\_TYPE [--connection\_type CONNECTION\_TYPE] [--connection\_data CONNECTION\_DATA] [--subsystem SUBSYSTEM]

Provision attached secure element with RSA keys

This example generates a complete set of RSA key files (\*.pem) (existing ones overwritten). Performs debug reset the attached secure element. Attached secure element provisioned with RSA key. Creates reference key from the injected RSA key.

### **optional arguments:**

**-h, --help** show this help message and exit

### **required arguments:**

**--key\_type KEY\_TYPE** Supported key types => rsa1024, rsa2048, rsa3072, rsa4096

### **optional arguments:**

**--connection\_type CONNECTION\_TYPE** Supported connection types => t1oi2c, sci2c, vcom, jrcpv1, jrcpv2, pcsc. Default: t1oi2c

**--connection\_data CONNECTION\_DATA** Parameter to connect to SE => eg. COM3, 127.0.0.1:8050, none. Default: none

**--subsystem SUBSYSTEM** Supported subsystem => se050, mbedtls. Default: se050

Example invocation:

```
python openssl_provisionRSA.py --key_type rsa1024
python openssl_provisionRSA.py --key_type rsa2048 --connection_data 169.254.0.1:8050
python openssl_provisionRSA.py --key_type rsa2048 --connection_data 127.0.0.1:8050 --
→connection_type jrcpv2
python openssl_provisionRSA.py --key_type rsa2048 --connection_data COM3
```

## openssl\_RSA.py

**usage:** openssl\_RSA.py [-h] --key\_type KEY\_TYPE [--connection\_data CONNECTION\_DATA]

Validation of OpenSSL Engine using RSA keys

This example showcases crypto operations and sign verify operations using RSA keys.

### optional arguments:

**-h, --help** show this help message and exit

### required arguments:

**--key\_type KEY\_TYPE** Supported key types => rsa1024, rsa2048, rsa3072, rsa4096

### optional arguments:

**--connection\_data CONNECTION\_DATA** Parameter to connect to SE => eg. COM3, 127.0.0.1:8050, none. Default: none

Example invocation:

```
python openssl_RSA.py --key_type rsa2048
python openssl_RSA.py --key_type rsa4096 --connection_data 127.0.0.1:8050
```

## rsa\_all.py

**usage:** rsa\_all.py [-h] [--connection\_data CONNECTION\_DATA] [--connection\_type CONNECTION\_TYPE] [--subsystem SUBSYSTEM]

Validation of OpenSSL Engine using RSA keys

This example injects keys with different supported RSA keys, then showcases Crypto & sign verify operations using those keys.

### optional arguments:

**-h, --help** show this help message and exit

**--connection\_data CONNECTION\_DATA** Parameter to connect to SE => eg. COM3, 127.0.0.1:8050, none. Default: none

**--connection\_type CONNECTION\_TYPE** Supported connection types => tloi2c, sci2c, vcom, jrcpv1, jrcpv2, pcsc. Default: tloi2c

**--subsystem SUBSYSTEM** Supported subsystem => se050. Default: se050

Example invocation:

```
python rsa_all.py
python rsa_all.py --connection_data 169.254.0.1:8050
python rsa_all.py --connection_data 127.0.0.1:8050 --connection_type jrcpv2
python rsa_all.py --connection_data COM3
```

## 6.2 Introduction on mbedTLS ALT Implementation

MbedTLS ALT implementation allows mbedTLS stack use the secure element access using SSS layer. Crypto operations performed during TLS handshake between client and server are performed using the secure element.

### 6.2.1 Using mbedTLS ALT

For reference, let's look at the `sss/ex/mbedtls/ex_sss_ssl2.c`. The important sections of the file are.

Here we initialize the keys and relevant objects.

```
/* pex_sss_demo_tls_ctx->obj will have the private key handle */
status = sss_key_object_init(&pex_sss_demo_tls_ctx->obj, &pCtx->ks);
if (status != kStatus_SSS_Success) {
 printf(" sss_key_object_init for keyPair Failed...\n");
 return kStatus_SSS_Fail;
}

status = sss_key_object_get_handle(&pex_sss_demo_tls_ctx->obj, SSS_KEYPAIR_INDEX_
 ↪CLIENT_PRIVATE);
if (status != kStatus_SSS_Success) {
 printf(" sss_key_object_get_handle for keyPair Failed...\n");
 return kStatus_SSS_Fail;
}

/* pex_sss_demo_tls_ctx->obj will have the private key handle */
status = sss_key_object_init(&pex_sss_demo_tls_ctx->pub_obj, &pCtx->ks);
if (status != kStatus_SSS_Success) {
 printf(" sss_key_object_init for Pub key Failed...\n");
 return kStatus_SSS_Fail;
}

/* pex_sss_demo_tls_ctx->obj will have the public key of counter part */
status = sss_key_object_get_handle(&pex_sss_demo_tls_ctx->pub_obj, SSS_PUBKEY_INDEX_
 ↪CA);
if (status != kStatus_SSS_Success) {
 printf(" sss_key_object_get_handle for extPubkey Failed...\n");
 return kStatus_SSS_Fail;
}

/* pex_sss_demo_tls_ctx->dev_cert will have the our device certificate */
status = sss_key_object_init(&pex_sss_demo_tls_ctx->dev_cert, &pCtx->ks);
if (status != kStatus_SSS_Success) {
 printf(" sss_key_object_init for Pub key Failed...\n");
 return kStatus_SSS_Fail;
}

status = sss_key_object_get_handle(&pex_sss_demo_tls_ctx->dev_cert, SSS_CERTIFICATE_
 ↪INDEX);
if (status != kStatus_SSS_Success) {
 printf(" sss_key_object_get_handle for client Cert Failed...\n");
 return kStatus_SSS_Fail;
}
```

Here, we tell mbedTLS to use the public key from the SE.

```
// for private key, we use the KEY from SE.
mbedtls_pk_free(&cacert.pk);
ret = sss_mbedtls_associate_pubkey(&cacert.pk, &pex_sss_demo_tls_ctx->pub_obj);
```

Here, get certificate in DER format from the SE, and then convert it to PEM and share it with the mbedtls stack.

```
size_t KeyBitLen = SIZE_CLIENT_CERTIFICATE * 8;
size_t KeyByteLen = SIZE_CLIENT_CERTIFICATE;

ret_code = sss_key_store_get_key(
 &pCtx->ks, &pex_sss_demo_tls_ctx->dev_cert, aclient_cer, &KeyByteLen, &KeyBitLen);

ret = mbedtls_x509_crt_parse_der(&clicert,
 (const unsigned char *)aclient_cer,
 sizeof(aclient_cer));
if ((ret_code == kStatus_SSS_Success) && (ret == 0)) {
 client_certificate_loaded = 1;
}
```

Here, we tell mbedtls to use the public key from the SE, generally for signing any contents.

```
sss_mbedtls_associate_keypair(&pkey, &pex_sss_demo_tls_ctx->obj);
```

Here, we tell mbedtls to use the public key from the SE for ECDH handshake.

```
sss_mbedtls_associate_ecdhctx(ssl.handshake, &pex_sss_demo_tls_ctx->obj, &pCtx->host_
 ↪ks);
```

## 6.2.2 Testing

### Building mbedtls SSL/DTLS server for testing

Build mbedtls server using the VS solution: CMake configurations: - RTOS\_Default: ON - WithHostCrypto\_MBEDTLS: ON - Withmbedtls\_ALT\_SSS: ON

- Project: mbedtls\_ex\_orig\_ssl\_server2/ex\_mbedtls\_origin\_dtls\_server

### Building mbedtls SSL/DTLS client (with SSS-APIs integration)

Build mbedtls client using the VS solution: CMake configurations: - RTOS\_Default: ON - WithHostCrypto\_MBEDTLS: ON - Withmbedtls\_ALT\_SSS: ON

- Project: mbedtls\_ex\_sss\_ssl2/mbedtls\_ex\_sss\_dtls

## Testings mbedTLS ALT

Directory `simw-top\sss\plugin\mbedtls\scripts` contains test scripts for starting mbedTLS server and client applications with different cipher suites. Before executing some test scripts, the secure element must first be provisioned.

- 1) Complete [Section 7.3 Steps needed before running ssscli tool](#)
- 2) Provision secure element using batch scripts in directory `simw-top\sss\plugin\mbedtls\scripts`. Run the following commands in virtual environment:

**To configure secure element for ECC** `windowsProvisionEC.bat <ec_curve> jrcpv2/vcom 127.0.0.1:8050/COM#`

**To configure secure element for RSA** `windowsProvisionRSA.bat <rsa_type> jrcpv2/vcom 127.0.0.1:8050/COM#`

**To see usage, run without any parameters** `windowsProvisionRSA.bat` or `windowsProvisionEC.bat`

---

**Note:** Once provisioning is done the virtual environment is not needed anymore.

---

- 3) Starting mbedTLS SSL client and server applications:

```
start_ssl2_server.bat <ec_curve>/<rsa_type>
start_ssl2_client.bat <ec_curve>/<rsa_type> <cipher suite> 127.0.0.1:8050/COM#
```

- 4) Starting mbedTLS DTLS client and server applications:

```
start_dtls_server.bat <ec_curve>/<rsa_type>
start_dtls_client.bat <ec_curve>/<rsa_type> <cipher suite> 127.0.0.1:8050/COM#
```

---

**Note:** Ensure that `ec_curve/rsa_type` used in server and client applications is the same as used while provisioning the SE in step 2.

---

## 6.2.3 mbedTLS ALT APIs

*group* **ax\_mbed\_tls**  
mbedTLS ALT implementation.

### Unnamed Group

int **sss\_mbedtls\_associate\_keypair** (mbedtls\_pk\_context \*pkey, *sss\_object\_t* \*pkeyObject)  
Associate a keypair provisioned in the secure element for subsequent operations.

**Return** 0 if successful, or 1 if unsuccessful

#### Parameters

- [out] pkey: Pointer to the `mbedtls_pk_context` which will be associated with data corresponding to the `key_index`
- [in] pkeyObject: The object that we are going to be use.

```
int sss_mbedtls_associate_pubkey (mbedtls_pk_context *pkey, sss_object_t *pkeyObject)
```

Associate a pubkey provisioned in the secure element for subsequent operations.

**Return** 0 if successful, or 1 if unsuccessful

**Parameters**

- [out] pkey: Pointer to the mbedtls\_pk\_context which will be associated with data corresponding to the key index
- [in] pkeyObject: The object that we are going to be use.

```
int sss_mbedtls_associate_ecdhctx (mbedtls_ssl_handshake_params *handshake,
 sss_object_t *pkeyObject, sss_key_store_t *hostKs)
```

Update ECDSA HandShake key with given indded.

**Return** 0 if successful, or 1 if unsuccessful

**Parameters**

- [inout] handshake: Pointer to the mbedtls\_ssl\_handshake\_params which will be associated with data corresponding to the key index
- [in] pkeyObject: The object that we are going to be use.
- [in] hostKs: Keystore to host for session key.

## 6.3 Platform Security Architecture

The Platform Security Architecture (PSA) by ARM is a holistic set of threat models, security analyses, hardware and firmware architecture specifications, and an open source firmware reference implementation

ARMmbed provides an interface for PSA APIs as a part of its mbed-crypto project. Also, it supports SE driver interface which allows the user to use its own implementation for PSA cryptographic functions. This is useful for integrating Secure Element with mbed-crypto provided PSA interface.

For details on PSA specification, refer to [ARMmbed PSA Specification](#).

### 6.3.1 PSA SE Driver Interface

The SE Driver interface allows the user to register Secure Element drivers for various cryptographic operations. It is not necessary that one driver should offer all cryptographic functionalities, we can register up to 4 drivers which may offer different functionalities.

Cryptographic APIs are split in to the following :

- SE\_KEY\_MANAGEMENT - Key management APIs like import/generate/destroy
- SE\_MAC - Mac operations
- SE\_CIPHER - Symmetric encrypt/decrypt operations
- SE\_AEAD - AEAD/GCM operations
- SE\_ASYMMETRIC - Asymmetric sign/verify/encrypt/decrypt operations
- SE\_KEY\_DERIVATION - Key derivation operations

The driver may support any subset of cryptographic functionalities, mbed-crypto offers software fall-back for APIs unavailable from SE.

---

**Note:** For SE interface, currently only SE\_KEY\_MANAGEMENT APIs, and asymmetric sign and asymmetric verify APIs are supported.

---

### 6.3.2 PSA Concepts

**Lifetime** The lifetime of an object refers to its persistence. An object can either be `PERSISTENT` or `VOLATILE`. However, for SE based PSA implementation, it refers to the module ID of SE Driver. Lifetime identifies the scope of an object (where the object is stored and which operations are available for it). This is used while performing any operation on an object. We can use different lifetimes while performing operations with the same object as long as different drivers can access the object.

**Slot ID** This is a 64-bit ID indicating where the object is loaded in the library. PSA offers a maximum of 31 slots, which indicates that at a time, a maximum of 31 objects can be used for any operation. If an object is not being used, the object handle can be closed to free the slot. Since there is no concept of an object being loaded for SE, the Slot ID will refer to the Key ID of the object. Any application will remain unaware of Slot ID and this should be managed internally. PSA library should use this value to refer to any object.

For SSS based PSA implementation, we have a 1:1 mapping of Slot ID from Key ID.

**Key ID** This is a 32-bit ID indicating the file name of the object which will be stored on the file system. Apart from the actual object, PSA also maintains an object file which will store details of the object such as policies and supported algorithms. Before performing any operation, these values are validated. The applications will use this value to refer to any object.

For SSS based PSA implementation, the contents of this file are also stored on SE.

#### PSA Objects

- **LIFETIME\_FILE** - This is SE specific file which can contain SE(driver) specific persistent data
- **TRANSACTION\_FILE** - This is a temporary file created at the time of any operation. It will be deleted after the operation.
- **OBJECT\_FILE** - This file corresponds to any object we create inside the SE. It will store data about policy, supported algorithms, etc. (keyID range is 0x20000000 - 0x2FFFFFFF)
- **OBJECT** - This is the actual object created inside the SE. (keyID range is 0x30000000 - 0x3FFFFFFF)

For any provided Key ID, the most significant nibble is masked out. The effective Key ID is 28-bit long.

**Warning:** This logic of managing KeyIDs of various PSA objects is temporary and it will be changed in the future.

### 6.3.3 Building PSA for TrustZone

PSA library is intended to run in ARM TrustZone. All examples will run in normal world and link to PSA library to perform cryptographic operations. Build the library for TrustZone with the following CMake configurations:

- Host=lpcpresso55s\_s
- HostCrypto=MBEDCRYPTO
- RTOS=Default
- SMCOM=T1oI2C
- PROJECT=PSA\_ALT

## 6.4 Android Key master

See *Android*

## 6.5 Introduction on Open62541 (OPC UA stack)

Open62541 is an open source C implementation of OPC UA stack. Open62541 supports binary encoding of messages with the following security profiles and modes.

Security Profiles

- 1) None
- 2) Basic256
- 3) Basic256SHA256

Security Modes

- 1) None
- 2) Sign
- 3) Sign and Encrypt

### 6.5.1 Integrating SE050 in Open62541

Open62541 stack uses mbedtls for all crypto operations in security profile plugins by default. For integrating with SE050, in the security profile plugins of Open62541 (`simw-top\ext\open62541\plugins\securityPolicies`), specific mbedtls functions used for private key operations, are replaced by calls to SSS APIs. Only the Basic256SHA256 security profile (uses RSA2048 keys) has been updated to support SE050 integration: private key operations - RSA Sign and RSA Decrypt - are now performed using SE050. The modified security profile plugin files are placed at `simw-top\sss\plugin\open62541`.

For the server application, a reference key file is used to pass the key id information to the security profile plugins layer.

---

**Note:** Please refer to the demonstrator built on top of the Open62541 OPC UA stack for further details (*OPC UA (Open62541) Demo*)

---



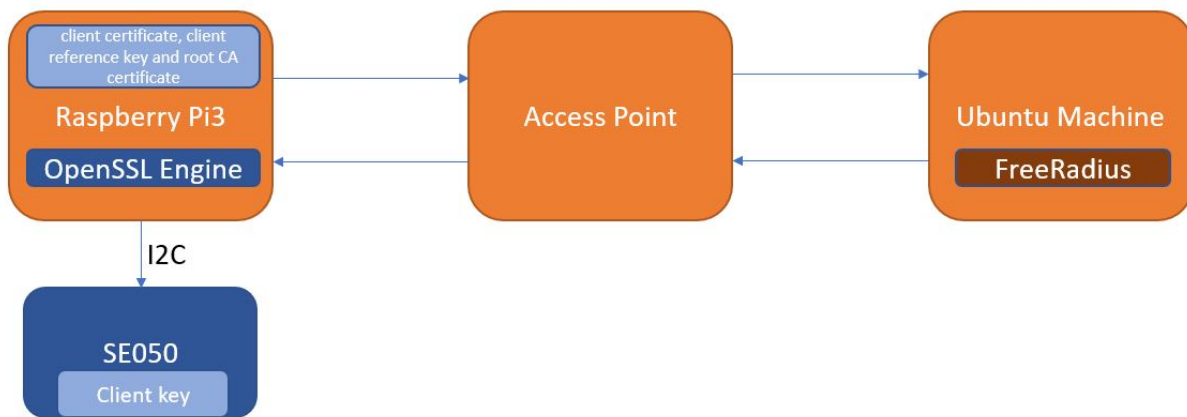
## 6.6 WiFi EAP Demo with Raspberry Pi3

### 6.6.1 Prerequisites

- Raspberry pi3 with raspbian OS installed.
- Ubuntu machine to run freeRadius
- Access point (WPA/WPA2 Enterprise capable)

### 6.6.2 Introduction

The image shows the wifi EAP demo set up



### 6.6.3 Setting up Access point

- 1) Connect Access point (WPA/WPA2 Enterprise capable) and Linux machine over a ethernet cable,
- 2) Log in to access point
- 3) Under wireless settings change security to WPA/WPA2 Enterprise and give the ip address of Ubuntu machine to RADIUS Server IP
- 4) RADIUS Server Port - 1812
- 5) Enter any password in RADIUS Server password field

## 6.6.4 Setting up freeradius Server on Ubuntu

- 1) **Install freeradius server on ubuntu machine.** `sudo apt-get install freeradius`

Freeradius is installed at /etc/freeradius

- 2) Now add the access point ip address and radius password to client configuration file - /etc/freeradius/clients.conf

```
For freeradius 2.2.8 add,
client 192.168.2.1/16 {
 secret = <radius server password mentioned in previous section>
 shortname = <Any short name>
}

For freeradius 3.0 add,
client router {
 ipaddr = 192.168.2.1
 secret = <radius server password mentioned in previous section>
}
```

- 3) Generate client and server keys. Scripts to generate keys and certificates are available in freeradius source code

```
git clone https://github.com/FreeRADIUS/freeradius-server.git
cd freeradius-server
```

- 4) **Add client email address and common name in /freeradius-server/raddb/certs/client.cnf**

```
41 prompt = no
42 distinguished_name = client
43 default_bits = 2048
44 input_password = whatever
45 output_password = whatever
46
47 [client]
48 countryName = FR
49 stateOrProvinceName = Radius
50 localityName = Somewhere
51 organizationName = Example Inc.
52 emailAddress = user3@example.org
53 commonName = user3
54
```

- 5) Execute bootstrap script at /freeradius-server/raddb/certs

```
./bootstrap
```

- 6) Keys and certificates are generated in folder /freeradius-server/raddb/certs

- 7) Copy root ca cert, server key and server certificate to installed freeradius path

```
cp /freeradius-server/raddb/certs/ca.pem /etc/freeradius/certs/
cp /freeradius-server/raddb/certs/server.key /etc/freeradius/certs/
```

(continues on next page)

(continued from previous page)

```
cp /freeradius-server/raddb/certs/server.pem /etc/freeradius/certs/
cp /freeradius-server/raddb/certs/dh /etc/freeradius/certs/
```

## 8) Add client details to user configuration file

For freeradius 2.2.8, add details in /etc/freeradius/users file

```
<user_name> Cleartext-password := <user_password>
Reply-Message = "<message>"
```

For freeradius 3.0 add details in /etc/freeradius/mods-config/files/authorize file

```
<user_name> Auth-Type := Accept, Cleartext-password := <user_password>
Reply-Message = "<message>"
```

## 9) Make the following changes to freeradius conf file at

For freeradius 2.2.8 - /etc/freeradius/eap.conf.

For freeradius 3.0 - /etc/freeradius/mods-available/eap.

```
@@ -199,6 +199,7 @@ eap {
 # *one* CA certificate.
 #
 # ca_file = /etc/ssl/certs/ca-certificates.crt
+ ca_file = /etc/freeradius/3.0/certs/ca.pem

 # OpenSSL will automatically create certificate chains,
 # unless we tell it to not do that. The problem is that
@@ -498,7 +499,7 @@ eap {
 #
 # You should also delete all of the files
 # in the directory when the server starts.
- # tmpdir = /tmp/radiusd
+ tmpdir = /tmp/radiusd

 # The command used to verify the client cert.
 # We recommend using the OpenSSL command-line
@@ -703,7 +704,8 @@ eap {
 # client certificate with EAP-TTLS, so this option is unlikely
 # to be usable for most people.
 #
- # require_client_cert = yes
+ EAP-TLS-Require-Client-Cert = Yes
+ require_client_cert = yes
}
```

## 10) Create a radiusd directory in /tmp and assign permission for freerad user

```
mkdir tmp/radiusd
sudo chown freerad:freerad tmp/radiusd
```

## 11) Start free radius server as

```
sudo freeradius -X
```

### 6.6.5 Setting up Raspberry Pi3

- 1) Copy plug and trust middleware package to rpi3 at /home/pi location
- 2) Modify the openssl engine id to pkcs11 in openssl engine header file ax\_embSeEngine.h.

Location: simw-top/sss/plugin/openssl/engine/inc/ax\_embSeEngine.h

- 3) Build openssl engine

```
cd simw-top
python scripts/create_cmake_projects.py
cd ../simw-top_build/raspbian_native_se050_tloi2c
make install
ldconfig /usr/local/lib
```

- 4) Copy client key (client.key), client certificate (client.crt), Root CA certificate (ca.pem) from ubuntu machine (freeradius-server/certs/) to raspberry pi at location /home/pi/wifiEAP
- 5) Refer to *CLI Tool* for pyCLI tool setup. Using pycli tool, create a reference pem file for client key

```
cd /home/pi/wifiEAP
openssl rsa -in client.key -out client.pem
sssccli connect se050 tloi2c none
sssccli set rsa pair 0x1234 client.pem
sssccli refpem rsa pair 0x1234 client_ref.pem
```

- 6) Add following network configuration to wpa\_supplicant.conf file (/etc/wpa\_supplicant)

```
pkcs11_engine_path=/usr/local/lib/libsss_engine.so
pkcs11_module_path=/usr/local/lib/libsss_engine.so

network={
 ssid="<SSID>"
 priority=1
 engine=1
 key_mgmt=WPA-EAP
 pairwise=CCMP TKIP
 auth_alg=OPEN
 eap=TTLS # When using freeradius 2.2.8, use TLS
 identity="<user_name>" # from user configuration file
 password="<user_password>" # from user configuration file

 ca_cert="/home/pi/wifiEAP/<ROOT_CA_CERT_FILE>"
 client_cert="/home/pi/wifiEAP/<CLIENT_CERT_FILE>"
 private_key="/home/pi/wifiEAP/<CLIENT_KEY_REFERENCE_FILE>"
 private_key_passwd="<PRIVATE_KEY_PASSWORD>" # If key file is not encrypted

 ca_cert2="/home/pi/wifiEAP/<ROOT_CA_CERT_FILE>"
 client_cert2="/home/pi/wifiEAP/<CLIENT_CERT_FILE>"
 private_key2="/home/pi/wifiEAP/<CLIENT_KEY_REFERENCE_FILE>"
 private_key_passwd="<PRIVATE_KEY_PASSWORD>" # If key file is not encrypted
}
```

- 7) Change the engine\_id to pkcs11 in openssl configuration file (/simw-top/demos/linux/common/openssl11\_sss\_se050.cnf)

```
[e4sss_se050_section]
engine_id = pkcs11
```

(continues on next page)

(continued from previous page)

```
dynamic_path = /usr/local/lib/libsss_engine.so
init = 1
default_algorithms = RAND, RSA, EC
```

8) Set the openssl config path as call:

```
$ export OPENSSL_CONF=/simw-top/demos/linux/common/openssl11_sss_se050.cnf
```

9) kill wpa\_supplicant process as

```
pkill wpa_supplicant
```

10) Restart wpa\_supplicant process as

```
wpa_supplicant -c/etc/wpa_supplicant/wpa_supplicant.conf -iwlan0 -Dwext
```

11) On successful TLS handshake, Rpi should be assigned with a valid IP address.

---

**Note:**

- 1) Tested with openssl version of 1.1.0j on raspberry pi.
  - 2) Ip address mentioned above is for illustrative purpose.
- 

## 6.7 PKCS#11 Standalone Library

PKCS#11 is a Public-Key Cryptography Standard for cryptographic data manipulation. It is mainly used with Hardware Security Modules and smart cards.

PKCS#11 standalone library is supported with SE05x for Linux based platforms.

### 6.7.1 Building on Linux/Raspberry Pi3

PKCS#11 standalone shared library can be built on Linux platforms and Raspberry Pi3.

Build PKCS#11 library for Raspberry pi 3 with the following CMake configurations:

- RTOS\_Default: ON
- WithHostCrypto\_MBEDTLS: ON
- Withmbedtls\_ALT\_SSS: ON
- Project: sss\_pkcs11

---

**Note:** The PKCS#11 library is not completely standalone as mbedtls library is also used for parsing data.

---

## 6.7.2 PKCS#11 specifications

*Token Label* SSS\_PKCS11

*Pin* Not required

*Supported Mechanisms*

- **RSA Mechanisms**
  - CKM\_RSA\_PKCS
  - CKM\_SHA1\_RSA\_PKCS
  - CKM\_SHA224\_RSA\_PKCS
  - CKM\_SHA256\_RSA\_PKCS
  - CKM\_SHA384\_RSA\_PKCS
  - CKM\_SHA512\_RSA\_PKCS
  - CKM\_RSA\_PKCS\_PSS
  - CKM\_SHA1\_RSA\_PKCS\_PSS
  - CKM\_SHA224\_RSA\_PKCS\_PSS
  - CKM\_SHA256\_RSA\_PKCS\_PSS
  - CKM\_SHA384\_RSA\_PKCS\_PSS
  - CKM\_SHA512\_RSA\_PKCS\_PSS
  - CKM\_RSA\_PKCS\_OAEP
- **AES Mechanisms**
  - CKM\_AES\_ECB
  - CKM\_AES\_CBC
  - CKM\_AES\_CTR
- **Digest Mechanisms**
  - CKM\_SHA\_1
  - CKM\_SHA224
  - CKM\_SHA256
  - CKM\_SHA384
  - CKM\_SHA512
- **ECDSA Mechanisms**
  - CKM\_ECDSA
  - CKM\_ECDSA\_SHA1
- **Key Generation Mechanisms**
  - CKM\_EC\_KEY\_PAIR\_GEN
  - CKM\_RSA\_PKCS\_KEY\_PAIR\_GEN
  - CKM\_AES\_KEY\_GEN
  - CKM\_DES2\_KEY\_GEN

- CKM\_DES3\_KEY\_GEN
- **Key Derivation Mechanisms**
- CKM\_ECDH1\_DERIVE

### 6.7.3 Using with pkcs11-tool

Generating new keypair:

```
pkcs11-tool --module $PKCS11_MODULE --keypairgen --key-type rsa:1024 --label_↵
↵ "sss:20202020"
```

Signing:

```
pkcs11-tool --module $PKCS11_MODULE --sign --label sss:20181001 -m SHA256-RSA-PKCS --
↵ slot 1 -i in.der -o signature.der
```

Decryption:

```
pkcs11-tool --module $PKCS11_MODULE --decrypt --label sss:20202020 -m SHA256-RSA-PKCS_↵
↵ --slot 1 -i in.der -o decrypt.der
```

Hashing:

```
pkcs11-tool --module $PKCS11_MODULE --hash -m SHA256 -i in.der -o hash.der
```

See also [Android](#)





## CLI TOOL

### 7.1 Introduction

The `sscli` tool is a Python based tool to manipulate the SE050 Secure Element.

It can be used to:

- Insert Keys, Certificates
- Generate Keys
- Attach policies to objects (To Be Done)
- Control Cloud for CLI (To Be Done)

It's written in Python to enable the tool and corresponding scripts to be run on Windows/Linux/OS X and other embedded devices. Because of Python language, the tool also enables us to generate complex provisioning scripts and schemas for testing various provisioning scenarios.

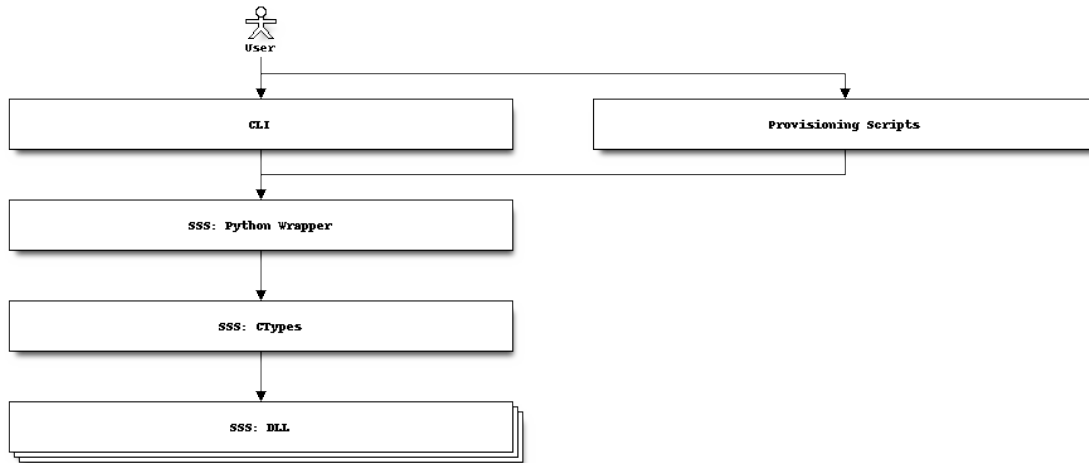
---

**Note:** Under Development

The CLI Tool is under development and under ALPHA State.

---

## 7.2 Block Diagram



| Name                 | Description                                                                                                                                                              |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| User                 | User who runs the scripts                                                                                                                                                |
| CLI                  | CLI Interface. Simple commands to interact with the SE.                                                                                                                  |
| Provisioning Scripts | Provisioning scripts for various use cases.                                                                                                                              |
| SSS: Python Wrapper  | Python middleware for SSS features. This is a functional abstraction layer over the raw CTypes wrapper.                                                                  |
| SSS: CTypes          | CTypes wrapper for SSS APIs. The wrapper is machine generated from SSS Header files.                                                                                     |
| SSS: DLL             | Native DLL on top of the SSS Layer. See <i>Plug &amp; Trust MW : Block Diagram</i> . Based on the selection of the platform/interface, this DLL needs to be re-compiled. |

## 7.3 Steps needed before running sssccli tool

### 7.3.1 Once per installation

The following steps are needed once per installation.

#### Windows

- 1) Ensure 32 bit PYTHON 3 is installed.  
You can download it from <https://www.python.org/downloads/>.
- 2) Environment to build host library is setup (GCC/MinGW/Visual Studio)

---

**Note:** This is required so that DLL/.SO can be prepared to be used by the CLI Tool.

---

- 3) Follow [Section 4.1 Windows Build](#) and build the sssapisw.

- 4) It is recommended to install the cli tool via virtualenv.

See <https://virtualenv.pypa.io/en/latest/> and <https://docs.python-guide.org/dev/virtualenvs/#lower-level-virtualenv> to understand more about virtualenv

Run:

```
pip3 install virtualenv
```

- 5) Once virtualenv is installed, create a new virtual environemnt:

```
python -m virtualenv venv
```

- 6) To activate the new created virtual ENV Run:

```
call venv\Scripts\activate.bat
```

- 7) **In the new installed virtualenv, install required packages. This includes click, cryptography and func-timeout.**

- click: To parse command line parameter.
- cryptography: Load keys, certificates and generate reference keys.
- func-timeout: ssscli recovery mechanism in case of no response from hardware.

Change directory to <SE05X\_root\_folder>/simw-top/pycli and run:

```
pip install -r requirements.txt
```

- 8) To install ssscli tool, run the following commands:

```
cd src
pip install --editable .
```

- 9) Alternately can install ssscli tool, by running following commands:

```
cd src
python setup.py develop
```

## IMX

- 1) Ensure PYTHON 3 is installed.

refer to [Section 9.5.1 platform-imx-linux](#)

- 2) Ensure func-timeout module is installed:

```
pip3 install func-timeout
```

- 3) Follow [Section 4.4 i.MX Linux Build](#) and build the sssapisw.

- 4) To install ssscli tool, change directory to <SE05X\_root\_folder>/simw-top/pycli/src and run the following command:

```
python3 setup.py develop
```

## Raspberry Pi

- 1) Ensure PYTHON 3 is installed.
- 2) Ensure python3-pip and libffi-dev are installed:

```
sudo apt-get install python3-pip
sudo apt-get install libffi-dev
```

- 3) Follow [Section 4.5 Raspberry Pi Build](#) and build the `ssapisw`.
- 4) Ensure click, cryptography and func-timeout modules are installed. To install these modules, change directory to `<SE05X_root_folder>/simw-top/pycli` and run the following command:

```
pip3 install -r requirements.txt
```

- 5) To install `sscli` tool, run the following commands:

```
cd src
sudo python3 setup.py develop
```

## 7.3.2 On change of interface

- 1) Native `sscli` DLL / `.so` needs to be built.

The DLL can be compiled with MSVC/MinGW/XCode/GCC based on the selected platform.

## 7.4 Running the `sscli` tool - Windows

- 1) To activate the new created virtual ENV Run:

```
call venv\Scripts\activate.bat
```

- 2) To connect to the SE, call:

```
sscli connect se050 vcom COM13
```

- 3) To dis-connect to the SE, call:

```
sscli disconnect
```

## 7.5 CLI Provisioning

### 7.5.1 Generating keys and certificates

For generating keys and certificates, following scripts generates using openssl.

- `GenerateAWSCredentials.py`
- `GenerateAZURECredentials.py`
- `GenerateGCPCredentials.py`
- `GenerateIBMCredentials.py`

The generated keys and certificates shall be available in `pycli/Provisioning/gcp`, `pycli/Provisioning/ibm`, `pycli/Provisioning/azure` and `pycli/Provisioning/aws` directories.

## 7.5.2 Provisioning for the demo

Generated keys and certificates are used to provision the secure element using `ResetAndUpdate_GCP.py`, `ResetAndUpdate_IBM.py`, `ResetAndUpdate_AZURE.py` and `ResetAndUpdate_AWS.py` scripts for gcp, ibm, azure and aws cloud demo respectively.

## 7.5.3 Steps to provision your device for demo on Windows

Provisioning on windows can be done in two ways.

- Using precompiled binaries
- Using python scripts

### Using precompiled binaries

Precompiled binaries available in `binaries/pySSSCLI` directory. Can generate certificates and provision the secure element by simply running these binaries.

- 1) For GCP, create certificate and provision, call:

```
Provision_GCP.exe <COM_PORT>
```

- 2) For IBM, create certificate and provision, call:

```
Provision_IBM.exe <COM_PORT>
```

- 3) For AWS, create certificate and provision, call:

```
Provision_AWS.exe <COM_PORT>
```

- 4) For AZURE, create certificate and provision, call:

```
Provision_AZURE.exe <COM_PORT>
```

The generated keys and certificates shall be available in `binaries/pySSSCLI/gcp`, `binaries/pySSSCLI/ibm`, `binaries/pySSSCLI/aws` and `binaries/pySSSCLI/azure` directories.

### Using Python scripts

- 1) Complete [Section 7.3 Steps needed before running ssscli tool](#)
- 2) from `pycli` directory, run:

```
call venv\Scripts\activate.bat
cd Provisioning
```

- 3) Check the vcom port number
- 4) For GCP, create certificate and provision, call:

```
python GenerateGCPCredentials.py <COM_PORT>
python ResetAndUpdate_GCP.py <COM_PORT>
```

- 5) For IBM, create certificate and provision, call:

```
python GenerateIBMCredentials.py <COM_PORT>
python ResetAndUpdate_IBM.py <COM_PORT>
```

- 6) For AWS, create certificate and provision, call:

```
python GenerateAWSCredentials.py <COM_PORT>
python verification_certificate.py <interCA_Certificate> <interCA_Keypair>
↪<verification_code>
python ResetAndUpdate_AWS.py <COM_PORT>
```

- 7) For AZURE, create certificate and provision, call:

```
python GenerateAZURECredentials.py <COM_PORT>
python verification_certificate.py <interCA_Certificate> <interCA_Keypair>
↪<verification_code>
python ResetAndUpdate_AZURE.py <COM_PORT>
```

- 8) Flash the demo on to the board

## 7.5.4 Steps to provision your device for demo on iMX or Raspberry Pi

- 1) Complete [Section 7.3 Steps needed before running ssscli tool](#)
- 2) from pycli directory, run:

```
cd Provisioning
```

- 3) For GCP, create certificate and provision, call:

```
python3 GenerateGCPCredentials.py
python3 ResetAndUpdate_GCP.py
```

- 4) For IBM, create certificate and provision, call:

```
python3 GenerateIBMCredentials.py
python3 ResetAndUpdate_IBM.py
```

- 5) For AWS, create certificate and provision, call:

```
python3 GenerateAWSCredentials.py
python3 verification_certificate.py <interCA_Certificate> <interCA_Keypair>
↪<verification_code>
python3 ResetAndUpdate_AWS.py
```

- 6) For AZURE, create certificate and provision, call:

```
python3 GenerateAZURECredentials.py
python3 verification_certificate.py <interCA_Certificate> <interCA_Keypair>
↪<verification_code>
python3 ResetAndUpdate_AZURE.py
```

- 7) Flash the demo on to the board

## 7.6 Usage Examples

### 7.6.1 SE050: VCOM Interface

Provisioning ECC Pair and Certificate:

```
ssscli connect se050 vcom COM5
ssscli se05x reset
ssscli set ecc pair 0x20181001 tstData\tls_client_key.pem
ssscli set cert 0x20181002 tstData\tls_client.cer
ssscli disconnect
```

Generating ecc key and retrieve public key:

```
ssscli connect se050 vcom COM5
ssscli se05x reset
ssscli generate ecc 0x20181006 NIST_P256
ssscli get ecc pair 0x20181006 data\tls_key.pem
ssscli disconnect
```

Inject and retrieve certificate:

```
ssscli connect se050 vcom COM5
ssscli set cert 0x20181004 tstData\tls_client.cer
ssscli get cert 0x20181004 data\extracted_certificate.cer
ssscli disconnect
```

Erase key, Inject ecc Key and Sign certificate:

```
ssscli connect se050 vcom COM5
ssscli erase 0x20181001
ssscli set ecc pair 0x20181001 tstData\tls_client_key.pem
ssscli sign 0x20181001 tstData\tls_client.cer data\signed_data.pem
ssscli disconnect
```

Inject and Retrieve AES key:

```
ssscli connect se050 vcom COM5
ssscli se05x reset
ssscli set aes 0x40100000 tstData\aes.der
ssscli get aes 0x40100000 data\extracted_aes_key.cer
ssscli disconnect
```

Inject ECC Public Key:

```
ssscli connect se050 vcom COM5
ssscli set ecc pub 0x20181010 tstData\tls_client_key_pub.pem
ssscli disconnect
```

Generate RSA Key and Get public key in DER format:

```
ssscli connect se050 vcom COM5
ssscli generate rsa 0x20182001 2048
ssscli get rsa pub 0x20182001 data\rsa_pub_2048.der --format DER
ssscli disconnect
```

Generate ecc Koblitz 256 Key, Sign Raw data and verify:

```
ssscli connect se050 vcom COM5
ssscli generate ecc 12E41001 Secp256k1
ssscli sign 12E41001 "Hello World" signed_data_ecc_secp256k1.pem
ssscli verify 12E41001 "Hello World" signed_data_ecc_secp256k1.pem
ssscli disconnect
```

Generate ecc Brainpool192 Key and Sign and verify certificate using SHA512 has algorithm:

```
ssscli connect se050 vcom COM5
ssscli generate ecc 0x2E101501 Brainpool192
ssscli sign 0x2E101501 tstData\tls_client.cer data\signed_data.pem --hashalgo=SHA512
ssscli verify 0x2E101501 tstData\tls_client.cer data\signed_data.pem --hashalgo=SHA512
ssscli disconnect
```

Read Cert UID of 10 bytes long:

```
ssscli connect se050 vcom COM5
ssscli se05x certuid
```

Read UID of 18 bytes long:

```
ssscli connect se050 vcom COM5
ssscli se05x uid
```

Session open with auth type as Platform SCP, generate ecc Brainpool192 Key:

```
ssscli connect se050 vcom COM5 --auth_type PlatformSCP
ssscli se05x reset
ssscli generate ecc 2E10D532 Brainpool192
ssscli disconnect
```

Generate ecc Koblitz256 key and create reference key:

```
ssscli connect se050 vcom COM5
ssscli generate ecc 7A10D838 Secp256k1
ssscli refpem ecc pair 7A10D838 data\refkey_secp256k1.pem
ssscli disconnect
```

Generate rsa 4096 key and create reference key in pkcs12 format:

```
ssscli connect se050 vcom COM5
ssscli generate rsa 0x70102040 4096
ssscli refpem rsa pair 0x70102040 rsa_4096_rekey.pfx --password nxp
ssscli disconnect
```

Generate ecc Brainpool 256 key and create pkcs12 format reference key extracted to pem format:

```
ssscli connect se050 vcom COM5
ssscli generate ecc 70102050 Brainpool256
ssscli refpem ecc pair 70102050 ecc_bp256_rekey.pem --format PKCS12 --password nxp
ssscli disconnect
```

Generate ecc ED25519 key and sign certificate:

```
ssscli connect se050 vcom COM5
ssscli generate ecc 70102060 ED25519
ssscli sign 70102060 tstData\tls_client.cer data\signed_data_using_ed25519.pem
ssscli disconnect
```



Generate ecc MONTH DH 25519 key:

```
ssscli connect se050 vcom COM5
ssscli generate ecc 70102080 ED25519
ssscli sign 70102080 tstData\tls_client.cer data\signed_data_using_ed25519.pem
ssscli disconnect
```

Perform Encrypt and Decrypt using RSA 2048:

```
ssscli connect se050 vcom COM5
ssscli generate rsa 0x20182001 2048
ssscli get rsa pub 0x20182001 rsa_pub_2048.pem
ssscli set rsa pub 0x20184120 rsa_pub_2048.pem
ssscli encrypt 0x20184120 "Welcome to NXP" rsa_2048_encrypted_data.pem
ssscli decrypt 0x20182001 rsa_2048_encrypted_data.pem decrypted_data.txt
ssscli disconnect
```

## 7.6.2 SE050: PCSC interface

Provisioning ECC Pair and Certificate:

```
ssscli connect se050 pcsc NXP
ssscli se05x reset
ssscli set ecc pair 0x20181001 tstData\tls_client_key.pem
ssscli set cert 0x20181002 tstData\tls_client.cer
ssscli disconnect
```

Inject ECC Public Key:

```
ssscli connect se050 pcsc NXP
ssscli set ecc pub 0x20181010 tstData\tls_client_key_pub.pem
ssscli disconnect
```

## 7.6.3 SE050: JRCPV2 interface

Provisioning ECC Pair and Certificate:

```
ssscli connect se050 jrctp2 127.0.0.1:8050
ssscli se05x reset
ssscli set ecc pair 0x20181001 tstData\tls_client_key.pem
ssscli set cert 0x20181002 tstData\tls_client.cer
ssscli disconnect
```

Generating ecc key and retrieve public key:

```
ssscli connect se050 jrctp2 127.0.0.1:8050
ssscli se05x reset
ssscli generate ecc 0x40100000 NIST_P256
ssscli get ecc pair 0x40100000 data\tls_key.pem
ssscli disconnect
```

Set and retrieve certificate:

```
ssscli connect se050 jrcpv2 127.0.0.1:8050
ssscli set cert 0x20181002 tstData\tls_client.cer
ssscli get cert 0x20181002 data\extracted_certificate.cer
ssscli disconnect
```

Erase a key, Inject ecc Key and Sign certificate:

```
ssscli connect se050 jrcpv2 127.0.0.1:8050
ssscli erase 0x20181001
ssscli set ecc pair 0x20181001 tstData\tls_client_key.pem
ssscli sign 0x20181001 tstData\tls_client.cer data\signed_data.pem
ssscli disconnect
```

Inject and Retrieve AES key:

```
ssscli connect se050 jrcpv2 127.0.0.1:8050
ssscli se05x reset
ssscli set aes 0x40200000 tstData\aes.der
ssscli get aes 0x40200000 data\extracted_aes_key.cer
ssscli disconnect
```

Inject ECC Public Key:

```
ssscli connect se050 jrcpv2 127.0.0.1:8050
ssscli set ecc pub 0x20181010 tstData\tls_client_key_pub.pem
ssscli disconnect
```

Generate RSA Key and Get public key in PEM format:

```
ssscli connect se050 jrcpv2 127.0.0.1:8050
ssscli generate rsa 0x20182001 2048
ssscli get rsa pub 0x20182001 data\rsa_pub_2048.pem --format PEM
ssscli disconnect
```

Generate ecc Koblitz 256 Key, Sign Raw data and verify:

```
ssscli connect se050 jrcpv2 127.0.0.1:8050
ssscli generate ecc 12E41001 Secp256k1
ssscli sign 12E41001 "Hello World" signed_data_ecc_secp256k1.pem
ssscli verify 12E41001 "Hello World" signed_data_ecc_secp256k1.pem
ssscli disconnect
```

Generate ecc Brainpool192 Key and Sign and verify certificate using SHA512 has algorithm:

```
ssscli connect se050 jrcpv2 127.0.0.1:8050
ssscli generate ecc 0x2E101501 Brainpool192
ssscli sign 0x2E101501 tstData\tls_client.cer data\signed_data.pem --hashalgo=SHA512
ssscli verify 0x2E101501 tstData\tls_client.cer data\signed_data.pem --hashalgo=SHA512
ssscli disconnect
```

Read Cert UID of 10 bytes long:

```
ssscli connect se050 jrcpv2 127.0.0.1:8050
ssscli se05x certuid
```

Read UID of 18 bytes long:

```
ssscli connect se050 jrcpv2 127.0.0.1:8050
ssscli se05x uid
```

Session open with auth type as Platform SCP, generate ecc Brainpool192 Key:

```
ssscli connect se050 jrcpv2 127.0.0.1:8050 --auth_type PlatformSCP
ssscli se05x reset
ssscli generate ecc 2E10D532 Brainpool192
ssscli disconnect
```

Generate ecc Koblitz256 key and create reference key:

```
ssscli connect se050 jrcpv2 127.0.0.1:8050
ssscli generate ecc 7A10D838 Secp256k1
ssscli refpem ecc pair 7A10D838 data\refkey_secp256k1.pem
ssscli disconnect
```

Generate rsa 4096 key and create reference key in pkcs12 format:

```
ssscli connect se050 jrcpv2 127.0.0.1:8050
ssscli generate rsa 0x70102040 4096
ssscli refpem rsa pair 0x70102040 rsa_4096_rekey.pfx --password nxp
ssscli disconnect
```

Generate ecc Brainpool 256 key and create pkcs12 format reference key extracted to pem format:

```
ssscli connect se050 jrcpv2 127.0.0.1:8050
ssscli generate ecc 70102050 Brainpool256
ssscli refpem ecc pair 70102050 ecc_bp256_rekey.pem --format PKCS12 --password nxp
ssscli disconnect
```

Generate ecc ED25519 key and sign certificate:

```
ssscli connect se050 jrcpv2 127.0.0.1:8050
ssscli generate ecc 70102060 ED25519
ssscli sign 70102060 tstData\tls_client.cer data\signed_data_using_ed25519.pem
ssscli disconnect
```

Generate ecc MONTH DH 25519 key:

```
ssscli connect se050 jrcpv2 127.0.0.1:8050
ssscli generate ecc 70102080 ED25519
ssscli sign 70102080 tstData\tls_client.cer data\signed_data_using_ed25519.pem
ssscli disconnect
```

Perform Encrypt and Decrypt using RSA 2048:

```
ssscli connect se050 jrcpv2 127.0.0.1:8050
ssscli generate rsa 0x20182001 2048
ssscli get rsa pub 0x20182001 rsa_pub_2048.pem
ssscli set rsa pub 0x20184120 rsa_pub_2048.pem
ssscli encrypt 0x20184120 "Welcome to NXP" rsa_2048_encrypted_data.pem
ssscli decrypt 0x20182001 rsa_2048_encrypted_data.pem decrypted_data.txt
ssscli disconnect
```

## 7.6.4 A71CH: VCOM Interface

Provisioning ECC Pair and Certificate:

```
ssscli connect a71ch vcom COM7
ssscli a71ch reset
ssscli set ecc pair 0x20181003 tstData\tls_client_key.pem
ssscli set cert 0x20181004 tstData\tls_client.cer
ssscli disconnect
```

Generating ecc key and retrieve public key:

```
ssscli connect a71ch vcom COM7
ssscli a71ch reset
ssscli generate ecc 0x20181003 NIST_P256
ssscli get ecc pair 0x20181003 data\tls_key.pem
ssscli disconnect
```

Set certificate and retrieve certificate:

```
ssscli connect a71ch vcom COM7
ssscli set cert 0x20181004 tstData\tls_client.cer
ssscli get cert 0x20181004 data\extracted_certificate.cer
ssscli disconnect
```

Erase a key, Inject ecc Key and Sign certificate:

```
ssscli connect a71ch vcom COM7
ssscli erase 0x20181005
ssscli set ecc pair 0x20181005 tstData\tls_client_key.pem
ssscli sign 0x20181005 tstData\tls_client.cer data\signed_data.pem
ssscli disconnect
```

## 7.6.5 A71CH: SCI2C interface

Provisioning ECC Pair and Certificate:

```
ssscli connect a71ch sci2c none
ssscli a71ch reset
ssscli set ecc pair 0x20181005 tstData/tls_client_key.pem
ssscli set cert 0x20181002 tstData/tls_client.cer
ssscli disconnect
```

Generating ecc key and retrieve public key:

```
ssscli connect a71ch sci2c none
ssscli a71ch reset
ssscli generate ecc 0x40100000 NIST_P256
ssscli get ecc pair 0x40100000 data/tls_key.pem
ssscli disconnect
```

Set certificate and retrieve certificate:

```
ssscli connect a71ch sci2c none
ssscli set cert 0x20181002 tstData/tls_client.cer
ssscli get cert 0x20181002 data/extracted_certificate.cer
ssscli disconnect
```

Erase a key, Inject ecc Key and Sign certificate:

```
ssscli connect a71ch sci2c none
ssscli erase 0x20181001
ssscli set ecc pair 0x20181001 tstData/tls_client_key.pem
ssscli sign 0x20181001 tstData/tls_client.cer data/signed_data.pem
ssscli disconnect
```

## 7.6 MBEDTLS

Provisioning ECC Pair and Certificate:

```
ssscli connect mbedtls none data
ssscli set ecc pair 0x20181005 tstData\tls_client_key.pem
ssscli set cert 0x20181002 tstData\tls_client.cer
ssscli disconnect
```

Generating ecc key and retrieve public key:

```
ssscli connect mbedtls none data
ssscli generate ecc 0x20181003 NIST_P256
ssscli get ecc pair 0x20181003 data\tls_key.pem
ssscli disconnect
```

Set certificate and retrieve certificate:

```
ssscli connect mbedtls none data
ssscli set cert 0x20181004 tstData\tls_client.cer
ssscli get cert 0x20181004 data\extracted_certificate.cer
ssscli disconnect
```

Erase key, provisioning ecc Key and Sign certificate:

```
ssscli connect mbedtls none data
ssscli erase 0x20181005
ssscli set ecc pair 0x20181005 tstData\tls_client_key.pem
ssscli sign 0x20181005 tstData\tls_client.cer data\signed_data.pem
ssscli disconnect
```

## 7.7 List of ssscli commands

SSSCLI uses PEM, DER and HEX data formats for keys and certificates. Refer *CLI Data formats*.

**Note:** Linux Environment

You can source `pycli/ssscli-bash-completion.sh` for auto-completion on bash with linux/posix based environemnt.

## 7.7.1 SSSCLI Commands

These are the top level commands accepted by the SSSCLI Tool.

1) ssscli:

```
Usage: ssscli [OPTIONS] COMMAND [ARGS]...

Command line interface for SE050

Options:
 -v, --verbose Enables verbose mode.
 --version Show the version and exit.
 --help Show this message and exit.

Commands:
 a71ch A71CH specific commands
 cloud (Not Implemented) Cloud Specific utilities.
 connect Open Session.
 decrypt Decrypt Operation
 disconnect Close session.
 encrypt Encrypt Operation
 erase Erase ECC/RSA/AES Keys or Certificate (contents)
 generate Generate ECC/RSA Key pair
 get Get ECC/RSA/AES Keys or certificates
 refpem Create Reference PEM/DER files (For OpenSSL Engine).
 se05x SE05X specific commands
 set Set ECC/RSA/AES Keys or certificates
 sign Sign Operation
 verify verify Operation
```

2) ssscli connect:

```
Usage: ssscli connect [OPTIONS] subsystem method port_name

Open Session.

subsystem = Security subsystem is selected to be used. Can be one of
"mbedtls, a71ch, se050, a71cl, openssl"

method = Connection method to the system. Can be one of "none, sci2c,
vcom, tloi2c, jrcpv1, jrcpv2, pcsc"

port_name = Subsystem specific connection parameters. Example: COM6,
127.0.0.1:8050. Use "None" where not applicable. e.g. SCI2C/Tloi2C

Options:
 --auth_type [None|PlatformSCP|UserID|ECKey|AESKey]
 Authentication type. Default is "None". Can
 be one of "None, UserID, ECKey, AESKey,
 PlatformSCP"
 --help Show this message and exit.
```

3) ssscli disconnect:

```
Usage: ssscli disconnect [OPTIONS]

Close session.
```

(continues on next page)

(continued from previous page)

```
Options:
 --help Show this message and exit.
```

#### 4) ssscli set:

```
Usage: ssscli set [OPTIONS] COMMAND [ARGS]...
```

**Set** ECC/RSA/AES Keys or certificates

```
Options:
 --help Show this message and exit.
```

Commands:

```
 aes Set AES Keys
 cert Set Certificate
 ecc Set ECC Keys
 hmac Set HMAC Keys
 rsa Set RSA Keys
```

#### 5) ssscli get:

```
Usage: ssscli get [OPTIONS] COMMAND [ARGS]...
```

Get ECC/RSA/AES Keys or certificates

```
Options:
 --help Show this message and exit.
```

Commands:

```
 aes Get AES Keys
 cert Get Certificate
 ecc Get ECC Keys
 rsa Get RSA Keys
```

#### 6) ssscli generate:

```
Usage: ssscli generate [OPTIONS] COMMAND [ARGS]...
```

Generate ECC/RSA Key pair

```
Options:
 --help Show this message and exit.
```

Commands:

```
 ecc Generate ECC Key
 pub Generate ECC Public Key to file
 rsa Generate RSA Key
```

#### 7) ssscli erase:

```
Usage: ssscli erase [OPTIONS] keyid
```

**Erase** ECC/RSA/AES Keys or Certificate (contents)

keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001

(continues on next page)

(continued from previous page)

```
Options:
 --help Show this message and exit.
```

**8) ssscli cloud:**

```
Usage: ssscli cloud [OPTIONS] COMMAND [ARGS]...

 (Not Implemented) Cloud Specific utilities.

 This helps to handle GCP/AWS/Watson specific settings.

Options:
 --help Show this message and exit.

Commands:
 aws (Not Implemented) AWS (Amazon Web Services) Specific utilities This...
 gcp (Not Implemented) GCP (Google Cloud Platform) Specific utilities
 This...
 ibm (Not Implemented) IBM Watson Specific utilities This helps to handle...
```

**9) ssscli a71ch:**

```
Usage: ssscli a71ch [OPTIONS] COMMAND [ARGS]...

 A71CH specific commands

Options:
 --help Show this message and exit.

Commands:
 reset Debug Reset A71CH
 uid Get A71CH Unique ID
```

**10) ssscli se05x:**

```
Usage: ssscli se05x [OPTIONS] COMMAND [ARGS]...

 SE05X specific commands

Options:
 --help Show this message and exit.

Commands:
 certuid Get SE05X Cert Unique ID (10 bytes)
 readidlist Read contents of SE050
 reset Reset SE05X
 uid Get SE05X Unique ID (18 bytes)
```

**11) ssscli refpem:**

```
Usage: ssscli refpem [OPTIONS] COMMAND [ARGS]...

 Create Reference PEM/DER files (For OpenSSL Engine).

Options:
```

(continues on next page)



(continued from previous page)

```
--help Show this message and exit.
```

Commands:

```
ecc Refpem ECC Keys
rsa Refpem RSA Keys
```

## 12) ssscli sign:

```
Usage: ssscli sign [OPTIONS] keyid certificate filename
```

Sign Operation

keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001

certificate = Input data to sign. can be raw data (DER format) or in file.  
**For** file, by default filename with extension .pem and .cer considered as PEM format and others as DER format.

filename = File name to store signature data. By default filename with extension .pem in PEM format and others in DER format.

Options:

```
--informat TEXT Input certificate format. TEXT can be "DER" or "PEM"
--outformat TEXT Output file format. TEXT can be "DER" or "PEM"
--hashalgo TEXT Hash algorithm. TEXT can be one of "SHA1, SHA224, SHA256,
 SHA384, SHA512,
 RSASSA_PKCS1_V1_5_SHA1,
 RSASSA_PKCS1_V1_5_SHA224,
 RSASSA_PKCS1_V1_5_SHA256,
 RSASSA_PKCS1_V1_5_SHA384,
 RSASSA_PKCS1_V1_5_SHA512,
 RSASSA_PKCS1_PSS_MGF1_SHA1,
 RSASSA_PKCS1_PSS_MGF1_SHA224,
 RSASSA_PKCS1_PSS_MGF1_SHA256,
 RSASSA_PKCS1_PSS_MGF1_SHA384,
 RSASSA_PKCS1_PSS_MGF1_SHA512"
--help Show this message and exit.
```

## 13) ssscli verify:

```
Usage: ssscli verify [OPTIONS] keyid certificate signature
```

**verify** operation

keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001

certificate = Input data for verification. Can be in file or raw data

**For** file, by default filename with extension .pem and .cer considered as PEM format and others as DER format.

filename = signature data file for verification. By default filename with extension .pem in PEM format and others in DER format.

Options:

```
--format TEXT certificate and signature file format. TEXT can be "DER" or
 "PEM"
```

(continues on next page)

(continued from previous page)

```
--hashalgo TEXT Hash algorithm. TEXT can be one of "SHA1, SHA224, SHA256,
 SHA384, SHA512,
 RSASSA_PKCS1_V1_5_SHA1,
 RSASSA_PKCS1_V1_5_SHA224,
 RSASSA_PKCS1_V1_5_SHA256,
 RSASSA_PKCS1_V1_5_SHA384,
 RSASSA_PKCS1_V1_5_SHA512,
 RSASSA_PKCS1_PSS_MGF1_SHA1,
 RSASSA_PKCS1_PSS_MGF1_SHA224,
 RSASSA_PKCS1_PSS_MGF1_SHA256, RSASSA_PKCS1_PSS_MGF1_SHA384,
 RSASSA_PKCS1_PSS_MGF1_SHA512"
--help Show this message and exit.
```

#### 14) ssscli encrypt:

```
Usage: ssscli encrypt [OPTIONS] keyid input_data filename

Sign Operation

keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001

input_data = Input data to Encrypt. can be raw string or in file.

filename = Output file name to store encrypted data. Encrypted data will
be stored in DER format.

Options:
--algo TEXT Algorithm. TEXT can be one of "oaep", "rsaes"
--help Show this message and exit.
```

#### 15) ssscli decrypt:

```
Usage: ssscli decrypt [OPTIONS] keyid encrypted_data filename

Sign Operation

keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001

encrypted_data = Encrypted data to Decrypt. can be raw data or in file.
Input data should be in DER format.

filename = Output file name to store Decrypted data.

Options:
--algo TEXT Algorithm. TEXT can be one of "oaep", "rsaes"
--help Show this message and exit.
```

## 7.7.2 Set Commands

These commands are used to set/put objects/keys to the target secure subsystem.

1) `ssscli set aes:`

```
Usage: ssscli set aes [OPTIONS] keyid key

Set AES Keys

keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001

key = Can be in file or raw key in DER or HEX format

Options:
--help Show this message and exit.
```

2) `ssscli set hmac:`

```
Usage: ssscli set hmac [OPTIONS] keyid key

Set HMAC Keys

keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001

key = Can be in file or raw key in DER or HEX format

Options:
--help Show this message and exit.
```

3) `ssscli set cert:`

```
Usage: ssscli set cert [OPTIONS] keyid key

Set Certificate

keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001

key = Can be raw certificate (DER format) or in file. For file, by default
filename with extension .pem and .cer considered as PEM format and others
as DER format.

Options:
--format TEXT Input certificate format. TEXT can be "DER" or "PEM"
--help Show this message and exit.
```

4) `ssscli set ecc pair:`

```
Usage: ssscli set ecc pair [OPTIONS] keyid key

Set ECC Key pair

keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001

key = Can be raw key (DER format) or in file. For file, by default
filename with extension .pem considered as PEM format and others as DER
format.
```

(continues on next page)

(continued from previous page)

```
Options:
 --format TEXT Input key format. TEXT can be "DER" or "PEM"
 --help Show this message and exit.
```

#### 5) ssscli set ecc pub:

```
Usage: ssscli set ecc pub [OPTIONS] keyid key

Set ECC Public Keys

keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001

key = Can be raw key (DER format) or in file. For file, by default
filename with extension .pem considered as PEM format and others as DER
format.

Options:
 --format TEXT Input key format. TEXT can be "DER" or "PEM"
 --help Show this message and exit.
```

#### 6) ssscli set rsa pair:

```
Usage: ssscli set rsa pair [OPTIONS] keyid key

Set RSA Key Pair

keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001

key = Can be raw key (DER format) or in file. For file, by default
filename with extension .pem considered as PEM format and others as DER
format.

Options:
 --format TEXT Input key format. TEXT can be "DER" or "PEM"
 --help Show this message and exit.
```

#### 7) ssscli set rsa pub:

```
Usage: ssscli set rsa pub [OPTIONS] keyid key

Set RSA Public Keys

keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001

key = Can be raw key (DER format) or in file. For file, by default
filename with extension .pem considered as PEM format and others as DER
format.

Options:
 --format TEXT Input key format. TEXT can be "DER" or "PEM"
 --help Show this message and exit.
```

### 7.7.3 Get Commands

These commands are used to retrieve/get objects/keys from the target secure subsystem.

1) `ssscli get aes:`

```
Usage: ssscli get aes [OPTIONS] keyid filename

Get AES Keys

keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001

filename = File name to store key. Data can be in PEM or DER format based
on file extension. By default filename with extension .pem in PEM format
and others in DER format.

Options:
--format TEXT Output file format. TEXT can be "DER" or "PEM"
--help Show this message and exit.
```

2) `ssscli get cert:`

```
Usage: ssscli get cert [OPTIONS] keyid filename

Get Certificate

keyid = 32bit Key ID. Should be in hex format. Example: 401286E6

filename = File name to store certificate. Data can be in PEM or DER
format based on file extension. By default filename with extension .pem
and .cer in PEM format and others in DER format.

Options:
--format TEXT Output file format. TEXT can be "DER" or "PEM"
--help Show this message and exit.
```

3) `ssscli get ecc pair:`

```
Usage: ssscli get ecc pair [OPTIONS] keyid filename

Get ECC Pair

keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001

filename = File name to store key. Data can be in PEM or DER format based
on file extension. By default filename with extension .pem in PEM format
and others in DER format.

Options:
--format TEXT Output file format. TEXT can be "DER" or "PEM"
--help Show this message and exit.
```

4) `ssscli get ecc pub:`

```
Usage: ssscli get ecc pub [OPTIONS] keyid filename

Get ECC Pub
```

(continues on next page)

(continued from previous page)

```
keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001

filename = File name to store key. Data can be in PEM or DER format based
on file extension. By default filename with extension .pem in PEM format
and others in DER format.

Options:
--format TEXT Output file format. TEXT can be "DER" or "PEM"
--help Show this message and exit.
```

#### 5) ssscli get rsa pair:

```
Usage: ssscli get rsa pair [OPTIONS] keyid filename

Get RSA Pair

keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001

filename = File name to store key. Data can be in PEM or DER format based
on file extension. By default filename with extension .pem in PEM format
and others in DER format.

Options:
--format TEXT Output file format. TEXT can be "DER" or "PEM"
--help Show this message and exit.
```

#### 6) ssscli get rsa pub:

```
Usage: ssscli get rsa pub [OPTIONS] keyid filename

Get RSA Pub

keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001

filename = File name to store key. Data can be in PEM or DER format based
on file extension. By default filename with extension .pem in PEM format
and others in DER format.

Options:
--format TEXT Output file format. TEXT can be "DER" or "PEM"
--help Show this message and exit.
```

## 7.7.4 Generate Commands

These commands are used to generate objects/keys inside the target secure subsystem.

#### 1) ssscli generate ecc:

```
Usage: ssscli generate ecc [OPTIONS] keyid [NIST_P192|NIST_P224|NIST_P256|NIST
_P384|NIST_P521|Brainpool160|Brainpool192|Brainpool
224|Brainpool256|Brainpool320|Brainpool384|Brainpoo
1512|Secp160k1|Secp192k1|Secp224k1|Secp256k1|ED_255
19|MONT_DH_25519|MONT_DH_448]

Generate ECC Key
```

(continues on next page)

(continued from previous page)

```
keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001

curvetype = ECC Curve type. can be one of "NIST_P192, NIST_P224,
NIST_P256, NIST_P384, NIST_P521, Brainpool160, Brainpool192, Brainpool224,
Brainpool256, Brainpool320, Brainpool384, Brainpool512, Secp160k1,
Secp192k1, Secp224k1, Secp256k1, ED_25519, MONT_DH_25519, MONT_DH_448"

Options:
--help Show this message and exit.
```

## 2) ssscli generate rsa:

```
Usage: ssscli generate rsa [OPTIONS] keyid [1024|2048|3072|4096]

Generate RSA Key

keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001

bits = Number of bits. can be one of "1024, 2048, 3072, 4096"

Options:
--help Show this message and exit.
```

## 7.7.5 Refpem Commands

These commands are used to get Reference/masked Keys usable by openssl engines.

### 1) ssscli refpem ecc pair:

```
Usage: ssscli refpem ecc pair [OPTIONS] keyid filename

Create reference PEM file for ECC Pair

keyid = 32bit Key ID. Should be in hex format. Example: 0x20E8A001

filename = File name to store key. Can be in PEM or DER or PKCS12 format
based on file extension. By default filename with extension .pem in PEM
format, .pfx or .p12 in PKCS12 format and others in DER format.

Options:
--format TEXT Output file format. TEXT can be "DER" or "PEM" or "PKCS12"
--password TEXT Password used for PKCS12 format.
--help Show this message and exit.
```

### 2) ssscli refpem ecc pub:

```
Usage: ssscli refpem ecc pub [OPTIONS] keyid filename

Create reference PEM file for ECC Pub

keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001

filename = File name to store key. Data Can be in PEM or DER format or
PKCS12 format based on file extension. By default filename with extension
```

(continues on next page)

(continued from previous page)

```
.pem in PEM format, .pfx or .p12 in PKCS12 format and others in DER
format.
```

Options:

```
--format TEXT Output file format. TEXT can be "DER" or "PEM" or "PKCS12"
--password TEXT Password used for PKCS12 format.
--help Show this message and exit.
```

### 3) ssscli refpem rsa pair:

```
Usage: ssscli refpem rsa pair [OPTIONS] keyid filename
```

Create reference PEM file for RSA Pair

keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001

filename = File name to store key. Data Can be in PEM or DER format or PKCS12 format based on file extension. By default filename with extension .pem in PEM format, .pfx or .p12 in PKCS12 format and others in DER format.

Options:

```
--format TEXT Output file format. TEXT can be "DER" or "PEM" or "PKCS12"
--password TEXT Password used for PKCS12 format.
--help Show this message and exit.
```

## 7.7.6 Se05x Commands

These are the SE05x specific commands.

### 1) ssscli se05x uid:

```
Usage: ssscli se05x uid [OPTIONS]
```

Get 18 bytes Unique ID from the SE05X Secure Module.

Options:

```
--help Show this message and exit.
```

### 2) ssscli se05x certuid:

```
Usage: ssscli se05x certuid [OPTIONS]
```

Get 10 bytes Cert Unique ID from the SE05X Secure Module. The cert uid is a subset of the Secure Module Unique Identifier

Options:

```
--help Show this message and exit.
```

### 3) ssscli se05x reset:

```
Usage: ssscli se05x reset [OPTIONS]
```

Resets the SE05X Secure Module to the initial state.

(continues on next page)



(continued from previous page)

This command uses ``Se05x\_API\_DeleteAll\_Iterative`` API of the SE05X MW to iterately delete objects provisioned inside the SE. Because of this, some objects are purposefully skipped from deletion.

It does not use the low level SE05X API ``Se05x\_API\_DeleteAll``

**For** more information, see documentation/implementation of the ``Se05x\_API\_DeleteAll\_Iterative`` API.

Options:

--help Show this message and exit.

#### 4) ssscli se05x readidlist:

Usage: ssscli se05x readidlist [OPTIONS]

Read contents of SE050

Options:

--help Show this message and exit.

## 7.7.7 A71CH Commands

These are the A71CH specific commands.

#### 1) ssscli a71ch uid:

Usage: ssscli a71ch uid [OPTIONS]

Get uid from the A71CH Secure Module.

Options:

--help Show this message and exit.

#### 2) ssscli a71ch reset:

Usage: ssscli a71ch reset [OPTIONS]

Resets the A71CH Secure Module to the initial state.

Options:

--help Show this message and exit.

## 7.8 CLI Data formats

### 7.8.1 DER

DER or [Distinguished Encoding Rules](#) files are digital certificates in binary format, instead of the ASCII PEM format. DER files may end with .der or .cer, so to differentiate between DER.cer and PEM.cer files, you may need to use a text editor to read the file. A DER file should not have any BEGIN/END statements and will show garbled binary content.

The file as seen in *hexl-mode* of emacs would look as below:

```

87654321 0011 2233 4455 6677 8899 aabb ccdd eeff 0123456789abcdef
00000000: 3081 8702 0100 3013 0607 2a86 48ce 3d02 0.....0...*.H.=.
00000010: 0106 082a 8648 ce3d 0301 0704 6d30 6b02 ...*.H.=....m0k.
00000020: 0101 0420 084f 2d70 eee4 09b9 5546 462eO-p....UFF.
00000030: 2cca 12e0 a046 90a3 3bea 8252 eeb5 4ff5 ,....F..;..R..O.
00000040: a4e9 b2d6 a144 0342 0004 633e 938a 67a7D.B..c>..g.
00000050: a9e6 f35f c369 3b51 3c88 6a0d c260 16f3 ..._.i;Q<.j..`..
00000060: 5e8c 2c0b 491b 2392 7af0 371d 28b1 1bfb ^.,.I.#.z.7.(...
00000070: e8f4 2e9c 5b0f 2897 c516 ec3e 7d62 97e4[.(....>}b..
00000080: c06f 936d ba9d ac8a bcd8 .O.m.....
--UU=:----F1 ecc_NIST_P256_prv_pkcs8.der All L1 (Hexl) -----

```

## 7.8.2 PEM

PEM or [Privacy Enhanced Mail](#) is a Base64 encoded DER certificate. PEM certificates are frequently used for web servers as they can easily be translated into readable data using a simple text editor. PEM file extension may contain .pem or .cer, so to differentiate when a PEM encoded file is opened in a text editor, it contains very distinct headers and footers.

Example:

```

-----BEGIN PRIVATE KEY-----
MIGHAgEAMBMGBYqGSM49AgEGCCqGSM49AwEHBG0wawIBAQQgCE8tcO7kCb1VRkYu
LMoS4KBGkKM76oJS7rVP9aTpstahRANCAARjPpOKZ6ep5vNfw2k7UTyIag3CYBbz
XowsC0kbI5J68DcdKLEb++j0LpxbDyiXxRbsPnlil+Tab5Ntup2sirzY
-----END PRIVATE KEY-----

```

## 7.8.3 HEX

Hex or [Hexadecimal](#) (base 16) is a positional system that represents numbers using a base of 16.

Example:

```

308187020100301306072a8648ce3d020106082a8648ce3d030107046d306b02
01010420084f2d70eee409b95546462e2cca12e0a04690a33bea8252eeb54ff5
a4e9b2d6a14403420004633e938a67a7a9e6f35fc3693b513c886a0dc26016f3
5e8c2c0b491b23927af0371d28b11bfbe8f42e9c5b0f2897c516ec3e7d6297e4
c06f936dba9dac8abcd8

```

## 7.8.4 REFERENCE KEY

- Refer to [Section 6.1.2 EC Reference key format](#) for EC reference key format.
- Refer to [Section 6.1.2 RSA Reference key format](#) for RSA reference key format.

## 7.9 Upload keys and certificates to SE05X using Pyccli tool

- 1) Refer to [CLI Tool](#) for pyCLI tool setup
- 2) Keys and Certificates can be uploaded to SE05X using the pyccli tool

```
ssscli connect se050 vcom <COMPORT>
ssscli se05x reset
ssscli set ecc pair <KEYID> path-to-key/<KEY_FILE>.pem
ssscli set cert <KEYID> path-to-certificate/<CERTIFICATE_FILE>.crt
ssscli disconnect
```

---

**Note:** ssscli se05x reset will delete all the existing keys and certificates

---



## 8.1 A71CH and SSS API

### 8.1.1 Introduction

The Plug&Trust Middleware provides support for the A71CH secure element through the SSS API. The full scope of the A71CH legacy API or the HLSE API is not covered by the SSS API. For use cases where this applies it's possible to use both SSS and A71CH API's (*Mixing SSS API and A71CH API*).

The A71CH support as included in the Plug&Trust Middleware, is derived from the A71CH Host Software package as available on [www.nxp.com/a71ch](http://www.nxp.com/a71ch). The `hostlib` directory contains refactored code that was previously published on [www.nxp.com/a71ch](http://www.nxp.com/a71ch).

This Plug&Trust Middleware provides the following additional functionality related to the A71CH:

- Compatibility with OpenSSL 1.1
- Support for the SSS API
- Cloud demos using SSS API
- OpenSSL Engine using SSS API

The following - as previously contained in the A71CH Host Software package - is no longer supported:

- Cloud demos using A71CH API (replaced by SSS API based cloud demos)

The SW build system is based upon cmake.

### 8.1.2 A71CH API to SSS API mapping

The following table provides an overview of the A71CH API's that can be replaced by SSS API's. As the usage of the SSS API is conceptually different from the A71CH API, there is no one-to-one replacement of API calls. Please consult *SSS APIs* for an introduction on using the SSS API and the applicable examples in *Section 5.2 SSS API Examples*.

The SSS Session concept - as applicable to A71CH - is restricted to establishing a connection between Host and Secure Element. Establishing an SCP03 session is orthogonal to the Session concept.

SSS specific policies are not applicable to A71CH.

| A71CH or HLSE API          | SSS equivalent available |                  |
|----------------------------|--------------------------|------------------|
| <b>a71ch_crypto_derive</b> |                          |                  |
| A71_HkdfExpandSymKey       | YES                      | sss_derive_key_* |

Continued on next page

Table 1 – continued from previous page

|                                     |     |                              |
|-------------------------------------|-----|------------------------------|
| A71_HkdfSymKey                      | YES | sss_derive_key_*             |
| A71_PskDeriveMasterSecret           | NO  |                              |
| A71_EcdhPskDeriveMasterSecret       | NO  |                              |
| A71_GetHmacSha256                   | YES | sss_mac_*                    |
| A71_HmacSha256Init                  | YES | sss_mac_*                    |
| A71_HmacSha256Update                | YES | sss_mac_*                    |
| A71_HmacSha256Final                 | YES | sss_mac_*                    |
|                                     |     |                              |
| <b>a71ch_crypto_ecc</b>             |     |                              |
| A71_GenerateEccKeyPair              | YES | sss_key_store_generate_key   |
| A71_GenerateEccKeyPairWithChallenge | NO  |                              |
| A71_GenerateEccKeyPairWithCode      | NO  |                              |
| A71_EccSign                         | YES | sss_asymmetric_sign_digest   |
| A71_EccNormalizedAsnSign            | NO  |                              |
| A71_EccRestrictedSign               | NO  |                              |
| A71_EccVerify                       | YES | sss_asymmetric_verify_digest |
| A71_EcdhGetSharedSecret             | YES | sss_derive_key_*             |
|                                     |     |                              |
| <b>a71ch_module</b>                 |     |                              |
| A71_GetCredentialInfo               | NO  |                              |
| A71_GetModuleInfo                   | NO  |                              |
| A71_GetUniqueID                     | YES | sss_session_prop_get_au8     |
| A71_GetCertUid                      | YES | sss_session_prop_get_au9     |
| A71_GetUnlockChallenge              | NO  |                              |
| A71_GetKeyPairChallenge             | NO  |                              |
| A71_GetPublicKeyChallenge           | NO  |                              |
| A71_GetRandom                       | YES | sss_rng_get_random           |
| A71_CreateClientHelloRandom         | NO  |                              |
| A71_GetRestrictedKeyPairInfo        | NO  |                              |
| A71_GetSha256                       | YES | sss_digest_one_go            |
| A71_Sha256Init/Update/Final         | YES | sss_digest_*                 |
| A71_InjectLock                      | NO  |                              |
| A71_LockModule                      | NO  |                              |
| A71_UnlockModule                    | NO  |                              |
| A71_SetTlsLabel                     | NO  |                              |
| A71_EccVerifyWithKey                | NO  |                              |
|                                     |     |                              |
| <b>a71ch_sst</b>                    |     |                              |
| A71_Erase_*_WithChallenge           | NO  |                              |
| A71_Erase_*_WithCode                | NO  |                              |
| A71_EraseEccKeyPair                 | YES | sss_key_store_erase_key      |
| A71_EraseEccPublicKey               | YES | sss_key_store_erase_key      |
| A71_EraseSymKey                     | NO  |                              |
| A71_Freeze_*_WithChallenge          | NO  |                              |
| A71_Freeze_*_WithCode               | NO  |                              |
| A71_FreezeEccKeyPair                | YES | sss_key_store_freeze_key     |
| A71_FreezeEccPublicKey              | YES | sss_key_store_freeze_key     |
| A71_FreezeGpData                    | NO  |                              |
| A71_FreezeSymKey                    | NO  |                              |
| A71_GetCounter                      | NO  |                              |

Continued on next page

Table 1 – continued from previous page

|                                |     |                       |
|--------------------------------|-----|-----------------------|
| A71_GetEccKeyPairUsage         | NO  |                       |
| A71_GetEccPublicKey            | YES | sss_key_store_get_key |
| A71_GetGpData                  | NO  |                       |
| A71_GetPublicKeyEccKeyPair     | YES | sss_key_store_get_key |
| A71_IncrementCounter           | NO  |                       |
| A71_SetConfigKey               | NO  |                       |
| A71_SetCounter                 | NO  |                       |
| A71_SetEccKeyPair              | YES | sss_key_store_set_key |
| A71_SetEccPublicKey            | YES | sss_key_store_set_key |
| A71_SetGpData                  | NO  |                       |
| A71_SetGpDataWithLockCheck     | NO  |                       |
| A71_SetRfc3394WrappedAesKey    | NO  |                       |
| A71_SetRfc3394WrappedConfigKey | NO  |                       |
| A71_SetSymKey                  | YES | sss_key_store_set_key |
| <b>HLSE</b>                    |     |                       |
| HLSE_GetObjectAttribute        |     |                       |
| HLSE_SetObjectAttribute        |     |                       |
| HLSE_EraseObject               |     |                       |
| HLSE_CreateObject              |     |                       |

### 8.1.3 Mixing SSS API and A71CH API

The Plug&Trust Middleware contains two examples illustrating how to use both the SSS API and the A71CH API from the same application.

#### ECC Example

The example uses the SSS API to sign and verify the digest. The example is available at `.../simw-top/demos/a71ch/ex_a71ch_sss_ecc.c`.

```

 status = sss_asymmetric_context_init(&ctx_asymm, &pCtx->session, &keyPair,
↪ kAlgorithm_SSS_SHA256, kMode_SSS_Sign);
 ENSURE_OR_GO_CLEANUP(status == kStatus_SSS_Success);

 signatureLen = sizeof(signature);
 /* Do Signing */
 LOG_I("Do Signing");
 LOG_MAU8_I("digest", digest, digestLen);
 status = sss_asymmetric_sign_digest(&ctx_asymm, digest, digestLen, signature, &
↪ signatureLen);
 ENSURE_OR_GO_CLEANUP(status == kStatus_SSS_Success);
 LOG_MAU8_I("signature", signature, signatureLen);
 LOG_I("Signing Successful !!!");
 sss_asymmetric_context_free(&ctx_asymm);

```

Next the example uses an A71CH API (A71\_GetPublicKeyEccKeyPair) to retrieve the public key from the A71CH. The A71CH specific key index is retrieved from the SSS object matching the key pair.

```

/* Access the A71CH with the (legacy) Host API */
SST_Index_t keyIdx = (((sss_sscp_object_t *)&keyPair)->slotId) & 0x0F;

```

(continues on next page)

(continued from previous page)

```

U8 pubEccKeyScratch[128];
U16 pubEccKeyScratchLen = 0;

LOG_I("A71_GetPublicKeyEccKeyPair(0x%02x)", keyIdx);
pubEccKeyScratchLen = sizeof(pubEccKeyScratch);
sw = A71_GetPublicKeyEccKeyPair(keyIdx, pubEccKeyScratch, &
↪pubEccKeyScratchLen);
status = ((sw == SW_OK) ? kStatus_SSS_Success : kStatus_SSS_Fail);

```

## AES key wrapping Example

The example uses the SSS API to set the AES key and the A71CH API to set the same AES key which is wrapped. Further to verify if the wrapped key is injected properly, a hkdf key is derived using both AES keys. The example is available at `.../simw-top/demos/a71ch/ex_a71ch_sss_aes_wrap_key`.

Injecting wrapped AES key starts with setting AES key which is used as KEK,

```

status = sss_key_object_init(&aesObj1, &pCtx->ks);
ENSURE_OR_GO_CLEANUP(status == kStatus_SSS_Success);

status = sss_key_object_allocate_handle(&aesObj1,
 MAKE_TEST_ID(__LINE__),
 kSSS_KeyPart_Default,
 kSSS_CipherType_AES,
 sizeof(aesKey),
 kKeyObject_Mode_Persistent);
ENSURE_OR_GO_CLEANUP(status == kStatus_SSS_Success);

status = sss_key_store_set_key(&pCtx->ks, &aesObj1, aesKey, sizeof(aesKey),
↪sizeof(aesKey) * 8, NULL, 0);
ENSURE_OR_GO_CLEANUP(status == kStatus_SSS_Success);

```

Now inject the wrapped AES key using the A71CH API - `A71_SetRfc3394WrappedAesKey`. Wrapped key length should be 24 bytes. Large keys can be set by calling the `A71_SetRfc3394WrappedAesKey` API multiple times and by incrementing the key index every time.

```

keyIdx = (((sss_sscp_object_t *)&aesObj1)->slotId) & 0x0F;

/* Set wrapped aes key - aesKey1 */
sw = A71_SetRfc3394WrappedAesKey(keyIdx, wapped_AesKey1_0, sizeof(wapped_
↪AesKey1_0));
status = ((sw == SW_OK) ? kStatus_SSS_Success : kStatus_SSS_Fail);
ENSURE_OR_GO_CLEANUP(status == kStatus_SSS_Success);

sw = A71_SetRfc3394WrappedAesKey(keyIdx + 1, wapped_AesKey1_1, sizeof(wapped_
↪AesKey1_1));
status = ((sw == SW_OK) ? kStatus_SSS_Success : kStatus_SSS_Fail);
ENSURE_OR_GO_CLEANUP(status == kStatus_SSS_Success);

```

Now verify if wrapped key injected is set correctly.

```

/* 1 - Calculate HKDF key with wrapped AES key injected - aesKey1 */
status = calculate_hkdf_key(pCtx, aesObj1, MAKE_TEST_ID(__LINE__), HkdfKey1, &
↪HkdfKey1Len);

```

(continues on next page)



(continued from previous page)

```

/* 2 - Inject aesKey1 AES key and calculate HKDF key */
status = sss_key_object_init(&aesObj2, &pCtx->ks);
ENSURE_OR_GO_CLEANUP(status == kStatus_SSS_Success);

status = sss_key_object_allocate_handle(&aesObj2,
 MAKE_TEST_ID(__LINE__),
 kSSS_KeyPart_Default,
 kSSS_CipherType_AES,
 sizeof(aesKey1),
 kKeyObject_Mode_Persistent);
ENSURE_OR_GO_CLEANUP(status == kStatus_SSS_Success);

status = sss_key_store_set_key(&pCtx->ks, &aesObj2, aesKey1, sizeof(aesKey1),
↪ sizeof(aesKey1) * 8, NULL, 0);
ENSURE_OR_GO_CLEANUP(status == kStatus_SSS_Success);

status = calculate_hkdf_key(pCtx, aesObj2, MAKE_TEST_ID(__LINE__), HkdfKey2, &
↪ HkdfKey2Len);

/* 3 - compare both hkdf keys generated */
if (0 != memcmp(HkdfKey1, HkdfKey2, HkdfKey1Len)) {
 status = kStatus_SSS_Fail;
}

```

## 8.1.4 SSS Object Identifier to A71CH Internal storage mapping

The SSS API uses a 32 bit unsigned value as key (object) identifier. The A71CH GP Storage contains the mapping between these key identifiers and A71CH internal storage as a dedicated data object of 160 byte.

The resulting A71CH KeyStore can contain upto:

- 4 ECC Key Pairs
- 3 ECC Public Keys
- 8 Symmetric Keys
- 4 Certificates

Any additional data object storage is only available through HLSE API calls (*A71CH Legacy HLSE (Generic) API*).

## 8.2 Miscellaneous

### 8.2.1 Demos and examples supported on A71CH

Refer to *DEMO List* to see the list of demo applications supported on A71CH. Make the following changes when testing with A71CH.

- 1) Set the Applet to A71CH and SMCOM to SCI2C in the build configuration and rebuild the middleware.
- 2) To provision A71CH for cloud application, change the *subsystem* to *a71ch* in *.../simw-top/pycli/src/Provision/Provision\_config.py* file.

```
SUBSYSTEM = "a71ch"
```

- 3) When testing cloud application on linux platform, set the OPENSSL\_CONF to A71CH specific openssl config files - openssl\_sss\_a71ch.cnf (for openssl 1.0) / openssl11\_sss\_a71ch.cnf (for openssl 1.1).

## 8.2.2 OpenSSL Engine

The Plug&Trust MW comes with two OpenSSL Engine implementations, both implementations support OpenSSL 1.1.1:

- SSS API based (A71CH SSS OpenSSL Engine)
- A71CH Legacy API based (A71CH Legacy OpenSSL Engine)

The reference key format and the tools supporting the reference keys are **different and incompatible**.

The implementation using the SSS API is documented in [Introduction on OpenSSL engine](#) and resides in `.../sss/plugin/openssl`. The functionality of the engine is restricted to EC NIST P-256 keys.

The implementation using the A71CH Legacy API resides in `.../hostlib/hostlib/embSeEngine`.

The reference key format used by the SSS OpenSSL Engine refers to the stored EC key by SSS Object Identifier. It relies upon the SSS Object Identifier to A71CH Internal storage mapping table ([A71CH and SSS API](#)) to locate the stored EC key in the attached A71CH.

The reference key format used by the A71CH Legacy OpenSSL Engine refers to the stored EC key by key class and key index. Both key class and index are specific to the A71CH secure element. The following provides an example of reference key format used by the A71CH Legacy OpenSSL Engine. The value reserved for the private key has been used to contain:

- a pattern of `0x10...00` to fill up the datastructure MSB side to the desired key length
- a 64 bit magic number (always `0xA5A6B5B6A5A6B5B6`)
- a byte (`0xkk`) to contain the key class (`0x10` for key pair and `0x20` for public key)
- a byte (`0xii`) to contain the key index (`0x00 to 0x03` for key pair and `0x00 to 0x02` for public key)

```
Private-Key: (256 bit)
priv:
 10:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:
 00:00:00:00:00:00:00:00:A5:A6:B5:B6:A5:A6:B5:B6:
 kk:ii
pub:
 04:1C:93:08:8B:26:27:BA:EA:03:D1:BE:DB:1B:DF:
 8E:CC:87:EF:95:D2:9D:FC:FC:3A:82:6F:C6:E1:70:
 A0:50:D4:B7:1F:F2:A3:EC:F8:92:17:41:60:48:74:
 F2:DB:3D:B4:BC:2B:F8:FA:E8:54:72:F6:72:74:8C:
 9E:5F:D3:D6:D4
ASN1 OID: prime256v1
```

### 8.2.3 A71CH and SCP03

Enabling SCP03 channel encryption on the A71CH is a two step process:

- [Phase-0] First the SCP03 keys must be set on the A71CH. The SCP03 keys can only be set once!
- [Phase-1] Once the SCP03 keys are set on the A71CH an SCP03 channel can be established between Host and A71CH. In case an SCP03 channel has been established successfully, the use of SCP03 becomes mandatory for all subsequent communication between Host and A71CH.

In the SSS API based example applications, two utility functions are used to support SCP03 channel encryption:

- `ex_a71ch_SetSeScp03Keys` is used to set the keys as required for [Phase-0]
- `SCP_Authenticate` is used to establish the SCP03 channel [Phase-1]

The example code (`sss/ex/inc/ex_sss_main_inc.h`) always combines these two steps and depends on the ‘Debug Reset’ command for this. In a product deployment the two phases must be distinct. [Phase-0] is only executed once. Ensure that the SCP03 keys are securely and persistently stored on the host.

To enable SCP03 in the SSS API examples one must set the following Cmake options:

```
-DA71CH_AUTH=SCP03
-DSCP=SCP03_HostCrypto
```

Please refer to [CMake Options](#) for more details and an overview of all available Cmake options.

**Note:** The Plug&Trust MW also contains example code illustrating the setting up of an SCP03 channel between Host and Secure Element for applications based upon the A71CH API: please refer to `hostlib/a71ch/ex/mainA71CH.c`

### 8.2.4 A71CH on Raspberry Pi

The default i2c master of Raspberry Pi doesn’t support the SMBUS ‘block read’ feature required for the sci2c protocol. As a workaround a software implementation of an i2c master must be used.

Add the following line to `/boot/config.txt` on the Raspberry Pi SD card and reboot:

```
dtoverlay=i2c-gpio,bus=4,i2c_gpio_delay_us=1,_i2c_gpio_sda=23,i2c_gpio_scl=24
```

This will create a `/dev/i2c-4` i2c port on Raspberry Pi.

Modify `.../simw-top/hostlib/hostLib/platform/rsp/i2c_a7.c` for correct i2c port

```
static char devName[] = "/dev/i2c-4";
```

The following table illustrates the connections to make between the Raspberry Pi Header and the A71CH.

Table 2: A71CH pin connections

| Raspberry Pi Header | A71CH     |
|---------------------|-----------|
| Pin# 1              | Power     |
| Pin# 6              | Ground    |
| Pin# 16             | I2C Data  |
| Pin# 18             | I2C Clock |

## 8.3 A71CH Legacy API

### 8.3.1 Introduction

The A71CH Legacy API encapsulates the APDU calls supported by the A71CH security module. The standard A71CH security module supports the following functionality:

- Secure storage, generation, insertion or deletion of ECC key pairs (ECC NIST P-256).
- Secure storage, insertion or deletion of ECC public keys.
- Signature generation and verification (ECDSA)
- Shared secret calculation for Key Agreement (ECDH or ECDH-E)
- Secure storage and use of monotonic counters (32 bits each)
- Secure storage, insertion or deletion of symmetric keys (128 bits); symmetric keys can be concatenated to form longer keys
- Retrieval of unique chip ID.
- HKDF using the symmetric secrets as key, Extract & Expand or Expand only.
- HMAC SHA256 calculation
- Freezing of credentials (= OTP behavior)
- An optional secure channel with the host MCU (conform Global Platform SCP03).

The Debug Mode variant of the A71CH security module, which can be ordered on evaluation kits, supports the following additional functionality:

- A set of debug commands to facilitate integration of the A71CH in a host application.
- Possibility to permanently disable these debug commands

---

**Note:** In the remainder of this document the A71CH Legacy API is simply called A71CH API

---

### 8.3.2 A71CH API

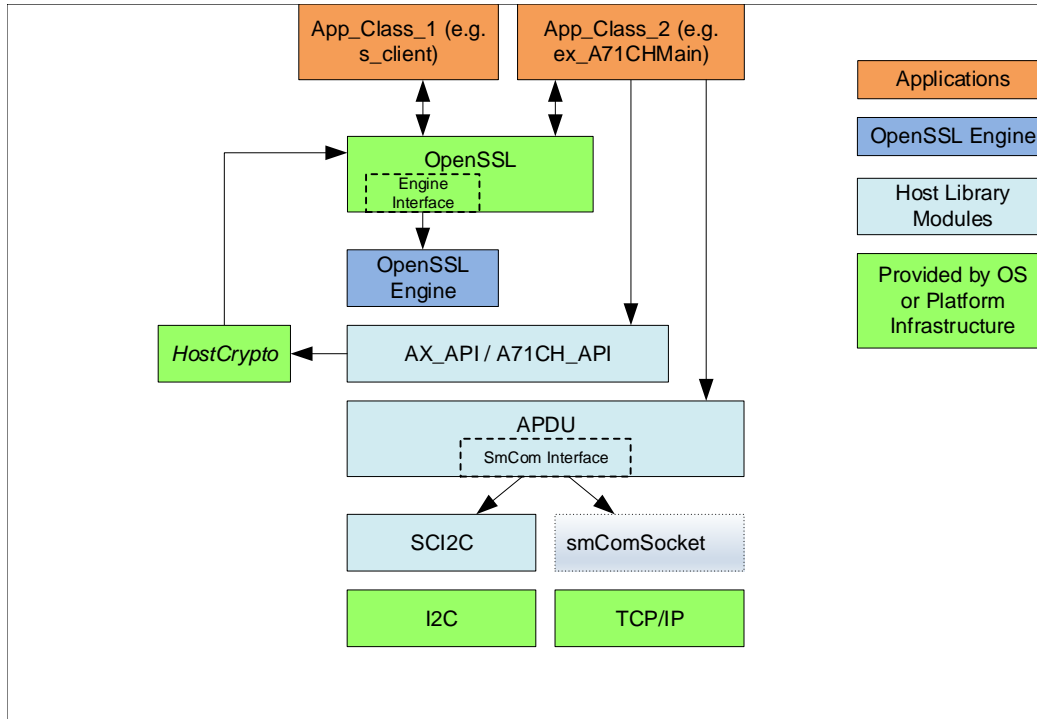
The A71CH API is made up of four parts:

- A71CH specific functionality (`.../hostlib/inc/a71ch_api.h`)
  - *Crypto Derive API* deals with deriving secrets, hmacs etc. from stored secrets
  - *Ecc Key API* deals with ECC crypto building blocks as ECDSA signing and verification and ECDH
  - *Module API* deals with functions not related to stored crypto credentials
  - *Secure Storage (SST) API* deals with storing, retrieving, erasing and locking credentials
- Data link communication functionality (`sm_connect.c`)
- Secure channel functionality (`ax_scp.h`). The implementation resides in `ax_scp.c` and `scp_a7x.c`.
- A71CH Debug Mode variant functionality (`a71_debug.c`)

### 8.3.3 SW structure

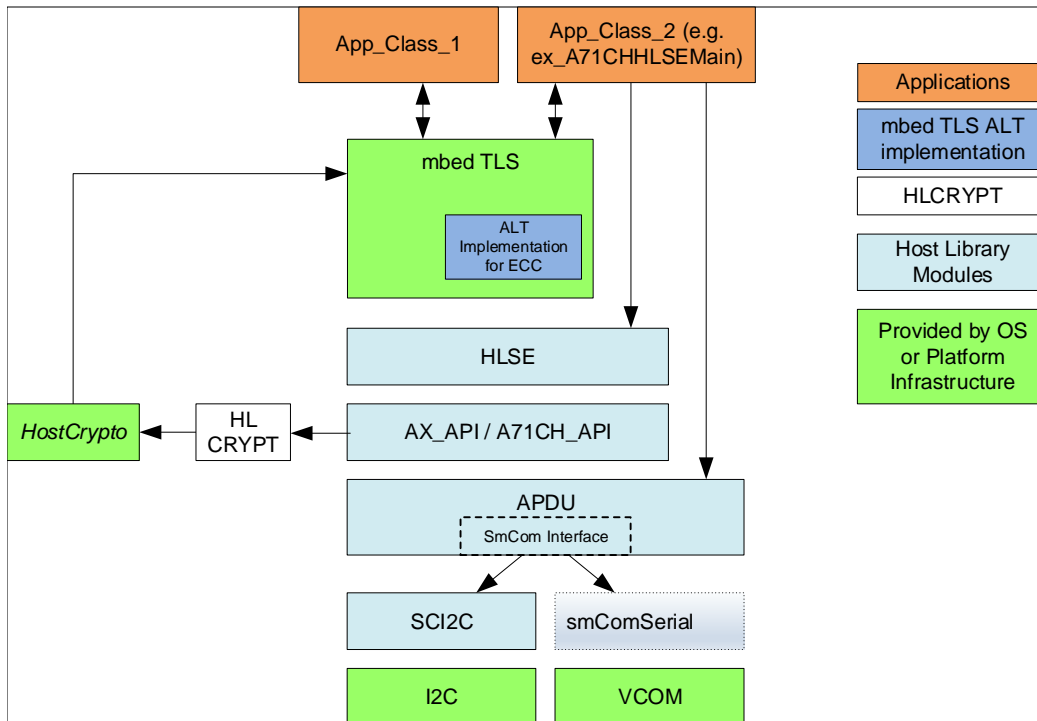
#### OpenSSL

The following picture illustrates the Host Library in the context of the Host SW with OpenSSL



#### MBED TLS

The following picture illustrates the Host Library in the context of the Host SW with mbed TLS



### 8.3.4 API details

#### Module API

**Description** Wrap module centric APDU functionality of the A71CH

#### Functions

U16 **A71\_GetCredentialInfo** (U8 \*map, U16 \*mapLen)

Get credential info from Module (in raw format)

##### Parameters

- [inout] map:
- [inout] mapLen:

##### Return Value

- SW\_OK: Upon successful execution

U16 **A71\_GetModuleInfo** (U16 \*selectResponse, U8 \*debugOn, U8 \*restrictedKpIdx, U8 \*transportLockState, U8 \*scpState, U8 \*injectLockState, U16 \*gpStorageSize)

Get info on Module

##### Parameters

- [out] selectResponse: Encodes applet revision and whether Debug Mode is available
- [out] debugOn: Equals 0x01 when the Debug Mode is available
- [out] restrictedKpIdx: Either the index of the restricted keypair or A71CH\_NO\_RESTRICTED\_KP

- [out] transportLockState: The value retrieved is one of A71CH\_TRANSPORT\_LOCK\_STATE\_LOCKED, A71CH\_TRANSPORT\_LOCK\_STATE\_UNLOCKED or A71CH\_TRANSPORT\_LOCK\_STATE\_ALLOW\_LOCK
- [out] scpState: The value retrieved is one of A71CH\_SCP\_MANDATORY, A71CH\_SCP\_NOT\_SET\_UP or A71CH\_SCP\_KEYS\_SET
- [out] injectLockState: The value retrieved is one of A71CH\_INJECT\_LOCK\_STATE\_LOCKED or A71CH\_INJECT\_LOCK\_STATE\_UNLOCKED
- [out] gpStorageSize: Total storage size (in byte) of the General Purpose data store

#### Return Value

- SW\_OK: Upon successful execution

U16 **A71\_GetUniqueID** (U8 \*uid, U16 \*uidLen)  
Get Unique Identifier from the Secure Module

#### Parameters

- [inout] uid: IN: buffer to contain uid; OUT: uid retrieved from Secure Module
- [inout] uidLen: IN: Size of buffer provided (at least A71CH\_MODULE\_UNIQUE\_ID\_LEN byte); OUT: length of retrieved unique identifier (expected to be A71CH\_MODULE\_UNIQUE\_ID\_LEN byte)

#### Return Value

- SW\_OK: Upon successful execution
- ERR\_WRONG\_RESPONSE: In case an identifier with a length different from A71CH\_MODULE\_UNIQUE\_ID\_LEN was retrieved

U16 **A71\_GetCertUid** (U8 \*certUid, U16 \*certUidLen)  
Get cert uid from the Secure Module. The cert uid is a subset of the Secure Module Unique Identifier

#### Parameters

- [inout] certUid: IN: buffer to contain cert uid; OUT: cert uid retrieved from Secure Module
- [inout] certUidLen: IN: Size of buffer provided (at least A71CH\_MODULE\_CERT\_UID\_LEN byte); OUT: length of retrieved unique identifier (expected to be A71CH\_MODULE\_CERT\_UID\_LEN byte)

#### Return Value

- SW\_OK: Upon successful execution
- ERR\_WRONG\_RESPONSE: In case the Secure Module Unique Identifier (i.e. the base uid) did not have the expected length

U16 **A71\_GetUnlockChallenge** (U8 \*challenge, U16 \*challengeLen)  
Get Unlock challenge from the Secure Module

#### Parameters

- [inout] challenge: IN: buffer to contain challenge; OUT: challenge retrieved from Secure Module
- [inout] challengeLen: IN: Size of buffer provided (at least A71CH\_MODULE\_UNLOCK\_CHALLENGE\_LEN byte); OUT: length of retrieved unique identifier (must be A71CH\_MODULE\_UNLOCK\_CHALLENGE\_LEN byte)

#### Return Value

- SW\_OK: Upon successful execution
- ERR\_WRONG\_RESPONSE: In case an identifier with a length different from A71CH\_MODULE\_UNLOCK\_CHALLENGE\_LEN was retrieved

U16 **A71\_GetKeyPairChallenge** (U8 \*challenge, U16 \*challengeLen)

Get Unlock challenge for a Keypair

**Parameters**

- [inout] challenge: IN: buffer to contain challenge; OUT: challenge retrieved from Secure Module
- [inout] challengeLen: IN: Size of buffer provided (at least A71CH\_MODULE\_UNLOCK\_CHALLENGE\_LEN byte); OUT: length of retrieved unique identifier (must be A71CH\_MODULE\_UNLOCK\_CHALLENGE\_LEN byte)

**Return Value**

- SW\_OK: Upon successful execution
- ERR\_WRONG\_RESPONSE: In case an identifier with a length different from A71CH\_MODULE\_UNLOCK\_CHALLENGE\_LEN was retrieved

U16 **A71\_GetPublicKeyChallenge** (U8 \*challenge, U16 \*challengeLen)

Get Unlock challenge for a Public Key

**Parameters**

- [inout] challenge: IN: buffer to contain challenge; OUT: challenge retrieved from Secure Module
- [inout] challengeLen: IN: Size of buffer provided (at least A71CH\_MODULE\_UNLOCK\_CHALLENGE\_LEN byte); OUT: length of retrieved unique identifier (must be A71CH\_MODULE\_UNLOCK\_CHALLENGE\_LEN byte)

**Return Value**

- SW\_OK: Upon successful execution
- ERR\_WRONG\_RESPONSE: In case an identifier with a length different from A71CH\_MODULE\_UNLOCK\_CHALLENGE\_LEN was retrieved

U16 **A71\_GetRandom** (U8 \*random, U8 randomLen)

Retrieves a random byte array of size randomLen from the Secure Module. The maximum amount of data that can be retrieved depends on whether an authenticated channel (SCP03) has been set up. In case SCP03 has been set up, this (worst-case) maximum is A71CH\_SCP03\_MAX\_PAYLOAD\_SIZE

**Parameters**

- [inout] random: IN: buffer to contain random value (at least of size randomLen); OUT: retrieved random data
- [in] randomLen: Amount of byte to retrieve

**Return Value**

- SW\_OK: Upon successful execution

U16 **A71\_CreateClientHelloRandom** (U8 \*clientHello, U8 clientHelloLen)

Updates a 32 byte random value inside the A71CH and returns this value to the caller.

**Post** A71CH is in a state it will accept A71\_PskDeriveMasterSecret or A71\_EcdhPskDeriveMasterSecret as an API call.

**Parameters**



- [inout] `clientHello`: IN: buffer to contain random value (at least of size `randomLen`); OUT: retrieved random data
- [in] `clientHelloLen`: Amount of byte to retrieve (must be equal to `AX_TLS_PSK_HELLO_RANDOM_LEN`)

**Return Value**

- `SW_OK`: Upon successful execution

U16 **A71\_GetRestrictedKeyPairInfo** (U8 \**idx*, U16 \**nBlocks*, U8 \**blockInfo*, U16 \**blockInfoLen*)

Get the index of the restricted key pair (*idx*) together with the number of modifiable blocks (*nBlocks*) in the locked GP storage area that is associated with the restricted key pair. Detailed info on block offset and block length is contained in the *blockInfo* byte array. Per block 2 bytes indicate the offset into GP storage and two bytes indicate the length of the modifiable block.

**Parameters**

- [out] *idx*: Index of restricted key pair. `A71CH_NO_RESTRICTED_KP` in case there is no restricted key pair
- [out] *nBlocks*: Number of modifiable blocks
- [inout] *blockInfo*: IN: Storage to contain *blockInfo*; OUT: Raw info on block offset and block length per block.
- [inout] *blockInfoLen*: IN: Size of *blockInfo* (in byte); OUT: effective size of *blockInfo*

**Return Value**

- `SW_OK`: Upon successful execution

U16 **A71\_GetSha256** (U8 \**data*, U16 *dataLen*, U8 \**sha*, U16 \**shaLen*)

Calculates the SHA256 value of the data provided as input.

**Parameters**

- [in] *data*: Data buffer for which the SHA256 must be calculated
- [in] *dataLen*: The length of data passed as argument
- [inout] *sha*: IN: caller passes a buffer of at least 32 byte; OUT: contains the calculated SHA256
- [inout] *shaLen*: IN: length of the sha buffer passed; OUT: because SHA256 is used this is 32 byte exact

**Return Value**

- `SW_OK`: Upon successful execution

U16 **A71\_Sha256Init** (void)

Initialise multistep SHA256.

**Return Value**

- `SW_OK`: Upon successful execution

U16 **A71\_Sha256Update** (U8 \**data*, U16 *dataLen*)

Update the data for calculating SHA256 value (in multistep).

**Parameters**

- [in] *data*: Data buffer for which the SHA256 must be calculated
- [in] *dataLen*: The length of data passed as argument

**Return Value**

- SW\_OK: Upon successful execution

U16 **A71\_Sha256Final** (U8 \*sha, U16 \*shaLen)  
calculating SHA256 value (in multistep).

**Parameters**

- [inout] sha: IN: caller passes a buffer of at least 32 byte; OUT: contains the calculated SHA256
- [inout] shaLen: IN: length of the sha buffer passed; OUT: because SHA256 is used this is 32 byte exact

**Return Value**

- SW\_OK: Upon successful execution

U16 **A71\_InjectLock** ()

This function disables - at device level - the ability to

- Set symmetric keys without prior wrapping
- Erase symmetric keys
- Set ECC key pairs (private key part) without prior wrapping
- Set ECC public key without prior wrapping

**Return Value**

- SW\_OK: Upon successful execution

U16 **A71\_LockModule** ()

This function locks the module (typically to protect the module during transport to production facilities). When the A71CH is locked the functionality is reduced to the following subset:

- A71\_GetUniqueID
- A71\_GetUnlockChallenge
- A71\_UnlockModule
- A71\_GetModuleInfo

**Return Value**

- SW\_OK: Upon successful execution

U16 **A71\_UnlockModule** (U8 \*code, U16 codeLen)

This function unlocks the module provided the correct code is provided as input argument. The A71CH can only be unlocked once: if the device is already unlocked, the device cannot be locked or unlocked again (it will remain unlocked).

The unlock code is calculated as follows:

- Request a challenge from A71CH using A71\_GetUnlockChallenge.
- Decrypt the challenge in ECB mode using the appropriate configuration key value (the same as stored at index A71CH\_CFG\_KEY\_IDX\_MODULE\_LOCK).
- The decrypted value is the unlock code

**Parameters**

- [in] code: Value of unlock code
- [in] codeLen: Length of unlock code (must be 16)

**Return Value**

- `SW_OK`: Upon successful execution

U16 **A71\_SetTlsLabel** (**const** U8 \**label*, U16 *labelLen*)

Sets the label that is used when calling `A71_EcdhPskDeriveMasterSecret` or `A71_PskDeriveMasterSecret`. Calling this function is optional. By default the label used by the A71CH is 'master secret' (no quotes) as applicable for TLS 1.2. The maximum size of the label that can be set is 24 byte.

**Parameters**

- [in] *label*: Value to be stored and used as 'label' in TLS 1.2 protocol
- [in] *labelLen*: Length of label (less than or equal to `A71CH_TLS_MAX_LABEL`)

**Return Value**

- `SW_OK`: Upon successful execution

U16 **A71\_EccVerifyWithKey** (**const** U8 \**pKeyData*, U16 *keyDataLen*, **const** U8 \**pHash*, U16 *hashLen*, **const** U8 \**pSignature*, U16 *signatureLen*, U8 \**pResult*)

Verifies whether *pSignature* is the signature of *pHash* using *pKeyData* as the verifying public key.

As opposed to function `A71_EccVerify` the public key value is passed as an argument to the A71CH.

**Parameters**

- [in] *pKeyData*: Public key passed as byte array in ANSI X9.62 uncompressed format
- [in] *keyDataLen*: Length of public key passed as argument
- [in] *pHash*: Pointer to the provided hash (or any other bytestring).
- [in] *hashLen*: Length of the provided hash.
- [in] *pSignature*: Pointer to the provided signature.
- [in] *signatureLen*: Length of the provided signature.
- [out] *pResult*: Pointer to the computed result of the verification. Points to a value of 0x01 in case of successful verification

**Return Value**

- `SW_OK`: Upon successful execution

**Ecc Key API**

**Description** Wrap the ECC cryptographic functionality of the A71CH.

**Functions**

U16 **A71\_GenerateEccKeyPair** (SST\_Index\_t *index*)

Generates an ECC keypair at storage location *index*.

**Pre** `INJECTION_LOCKED` has not been set

**Parameters**

- [in] *index*: Storage index of the keypair to be created.

**Return Value**

- `SW_OK`: Upon successful execution

U16 **A71\_GenerateEccKeyPairWithChallenge** (SST\_Index\_t *index*, **const** U8 \**configKey*, U16 *configKeyLen*)

Generates an ECC keypair at storage location *index*. This function must be called instead of A71\_GenerateEccKeyPair in case INJECTION\_LOCKED was set.

To use this function the value of the Key Pair configuration key must be known on the host. If this is not the case use A71\_GenerateEccKeyPairWithCode instead.

#### Parameters

- [in] *index*: Storage index of the keypair to be created.
- [in] *configKey*: Value of Key Pair configuration key. This value has a high level of confidentiality and may not be available to the Host.
- [in] *configKeyLen*: Length of Key Pair configuration key

#### Return Value

- SW\_OK: Upon successful execution

U16 **A71\_GenerateEccKeyPairWithCode** (SST\_Index\_t *index*, **const** U8 \**code*, U16 *codeLen*)

Generates an ECC keypair at storage location *index*. This function must be called instead of A71\_GenerateEccKeyPair in case INJECTION\_LOCKED was set.

The assumption is the value of the Key Pair configuration key is not known on the host. If this does not apply use A71\_GenerateEccKeyPairWithChallenge instead.

The code is calculated as follows:

- Request a challenge from A71CH using A71\_GetUnlockChallenge.
- Decrypt the challenge in ECB mode using the appropriate configuration key value (the same as stored at `index::A71XX_CFG_KEY_IDX_PRIVATE_KEYS`).
- The decrypted value is the value of *code*

#### Parameters

- [in] *index*: Storage index of the keypair to be created.
- [in] *code*: Value of unlock code.
- [in] *codeLen*: Length of unlock code (must be 16)

#### Return Value

- SW\_OK: Upon successful execution

U16 **A71\_EccSign** (SST\_Index\_t *index*, **const** U8 \**pHash*, U16 *hashLen*, U8 \**pSignature*, U16 \**pSignatureLen*)

Signs the hash *pHash* using the keypair at the indicated index.

#### Parameters

- [in] *index*: Storage index of the keypair (private key) to be used.
- [in] *pHash*: Pointer to the provided hash (or any other bytestring).
- [in] *hashLen*: Length of the provided hash.
- [inout] *pSignature*: Pointer to the computed signature.
- [inout] *pSignatureLen*: Pointer to the length of the computed signature.

#### Return Value

- SW\_OK: Upon successful execution

U16 **A71\_EccNormalizedAsnSign** (SST\_Index\_t *index*, **const** U8 \**pHash*, U16 *hashLen*, U8 \**pSignature*, U16 \**pSignatureLen*)

Signs the hash *pHash* using the keypair at the indicated index.

The integer representation of the ECDSA signatures' r and s component is modified to be in line with ASN.1 (Ensuring an integer value is always encoded in the smallest possible number of octets)

#### Parameters

- [in] *index*: Storage index of the keypair (private key) to be used.
- [in] *pHash*: Pointer to the provided hash (or any other bytestring).
- [in] *hashLen*: Length of the provided hash.
- [inout] *pSignature*: Pointer to the computed signature.
- [inout] *pSignatureLen*: Pointer to the length of the computed signature.

#### Return Value

- SW\_OK: Upon successful execution

U16 **A71\_EccRestrictedSign** (SST\_Index\_t *index*, **const** U8 \**updateBytes*, U16 *updateBytesLen*, U8 \**invocationCount*)

Patches a predetermined fixed size memory region in GP storage with the byte array *updateBytes* Creates a signed certificate - in place in GP storage - using a predetermined block of GP storage data

#### Parameters

- [in] *index*: Storage index of the keypair (private key) to be used.
- [in] *updateBytes*: Byte array to be written into GP storage
- [in] *updateBytesLen*: Length of the provided byte array (*updateBytes*).
- [out] *invocationCount*: Amount of times the underlying APDU has been called successfully.

#### Return Value

- SW\_OK: Upon successful execution

U16 **A71\_EccVerify** (SST\_Index\_t *index*, **const** U8 \**pHash*, U16 *hashLen*, **const** U8 \**pSignature*, U16 \**signatureLen*, U8 \**pResult*)

Verifies whether *pSignature* is the signature of *pHash* using the public key stored under *index* as the verifying public key.

The index refers to an instance of the PUBLIC\_KEY secure storage class on the A71CH.

**Note** The public key of an ECC key pair cannot be used for a verify operation.

**Note** A71\_EccVerifyWithKey allows to pass the value of the public key rather than use a stored public key.

#### Parameters

- [in] *index*: Storage index of the key used for the verification.
- [in] *pHash*: Pointer to the provided hash (or any other bytestring).
- [in] *hashLen*: Length of the provided hash (*pHash*).
- [in] *pSignature*: Pointer to the provided signature.
- [in] *signatureLen*: Length of the provided signature (*pSignature*)

- [out] pResult: Pointer to the computed result of the verification. Points to a value of 0x01 in case of successful verification

#### Return Value

- SW\_OK: Upon successful execution

U16 **A71\_EcdhGetSharedSecret** (U8 *index*, **const** U8 \**pOtherPublicKey*, U16 *otherPublicKeyLen*, U8 \**pSharedSecret*, U16 \**pSharedSecretLen*)

Generates and retrieves a shared secret ECC point pSharedSecret using the private key stored at index and a public key pOtherPublicKey passed as argument.

#### Parameters

- [in] index: to the key pair (private key to be used)
- [in] pOtherPublicKey: Pointer to the given public key.
- [in] otherPublicKeyLen: Length of the given public key.
- [inout] pSharedSecret: Pointer to the computed shared secret.
- [inout] pSharedSecretLen: Pointer to the length of the computed shared secret.

#### Return Value

- SW\_OK: Upon successful execution

## Crypto Derive API

**Description** Wrap the key derivation functionality of the A71CH.

### Functions

U16 **A71\_HkdfExpandSymKey** (SST\_Index\_t *index*, U8 *nBlock*, **const** U8 \**info*, U16 *infoLen*, U8 \**derivedData*, U16 *derivedDataLen*)

The HMAC Key Derivation function derives a key from a stored secret using SHA256 as hash function according to [RFC5869]. Only the expand step will be executed.

The secret is stored in the SYM key store. It can be either 16, 32, 48 or 64 byte long. The Most Significant part of the secret resides in the storage location with the lowest index. The subsequent parts reside in the next storage locations. The nBlock parameter is equal to the length of the secret divided by 16. A secret with length 64 can only start at Index 0 of the SYM key store: a secret can not be stored wrapped around in the SYM key store.

**Note** infoLen must be smaller than 254 byte.

#### Parameters

- [in] index: Index of the SYM key store containing the MSB part of the pre-shared secret
- [in] nBlock: Amount of blocks, equivalent to the pre-shared secret length when multiplied by 16
- [in] info: Context and application specific information used in expand step
- [in] infoLen: The length of the info data passed as argument
- [inout] derivedData: IN: caller passes a buffer of at least derivedDataLen; OUT: contains the calculated derived data
- [in] derivedDataLen: IN: length of the requested derivedData. Must be smaller than 256 byte.

#### Return Value

- SW\_OK: Successfull execution

U16 **A71\_HkdfSymKey** (SST\_Index\_t *index*, U8 *nBlock*, **const** U8 *\*salt*, U16 *saltLen*, **const** U8 *\*info*, U16 *infoLen*, U8 *\*derivedData*, U16 *derivedDataLen*)

The HMAC Key Derivation function derives a key from a stored secret using SHA256 as hash function according to [RFC5869]. Both the extract and expand steps will be executed.

In case a zero length salt value is passed as argument, this function is equivalent to A71\_HkdfExpandSymKey: i.e. the extract step is skipped. To enforce the usage of the default salt value (a Bytestring of 32 zeroes) the caller must explicitly pass this default salt value as argument to this function.

The secret is stored in the SYM key store. It can be either 16, 32, 48 or 64 byte long. The Most Significant part of the secret resides in the storage location with the lowest index. The subsequent parts reside in the next storage locations. The nBlock parameter is equal to the length of the secret divided by 16. A secret with length 64 can only start at Index 0 of the SYM key store: a secret can not be stored wrapped around in the SYM key store.

**Note** The sum of saltLen and infoLen must be smaller than 254 byte.

#### Parameters

- [in] *index*: Index of the SYM key store containing the MSB part of the pre-shared secret
- [in] *nBlock*: Amount of blocks, equivalent to the pre-shared secret length when multiplied by 16
- [in] *salt*: Salt data used in extract step
- [in] *saltLen*: The length of the salt data passed as argument
- [in] *info*: Context and application specific information used in expand step
- [in] *infoLen*: The length of the info data passed as argument
- [inout] *derivedData*: IN: caller passes a buffer of at least derivedDataLen; OUT: contains the calculated derived data
- [in] *derivedDataLen*: IN: length of the requested derivedData. Must be smaller than 256 byte.

#### Return Value

- SW\_OK: Successfull execution

U16 **A71\_PskDeriveMasterSecret** (SST\_Index\_t *index*, U8 *nBlock*, **const** U8 *\*serverHelloRnd*, U16 *serverHelloRndLen*, U8 *\*masterSecret*)

This function calculates the PRF according to TLS1.2 [RFC5246]. The pre-master secret is formed - based upon a pre-shared secret (PSK) stored in the secure module - according to [RFC4279].

The pre-shared secret is stored in the SYM key store. It can be either 16, 32, 48 or 64 byte long. The Most Significant part of the pre-shared secret resides in the storage location with the lowest index. The subsequent parts reside in the next storage locations. The nBlock parameter is equal to the length of the PSK divided by 16.

A PSK cannot be stored wrapped around in the SYM key store.

The PRF creating the masterSecret also takes as parameter the concatenation of label ("master\_secret"), ClientHello.random and ServerHello.random. This function only takes ServerHello.random as parameter: ClientHello.random has already been set by a call to A71\_CreateClientHelloRandom, the value of the label (default is "master\_secret") can be overruled by a call to A71\_SetTlsLabel.

**Pre** This call must be preceded by a call to A71\_CreateClientHelloRandom, no other A71CH API call (implying an APDU exchange between Host and A71CH) may be executed in between the invocation of A71\_CreateClientHelloRandom and A71\_PskDeriveMasterSecret

#### Parameters

- [in] `index`: Index of the SYM key store containing the MSB part of the pre-shared secret
- [in] `nBlock`: Amount of blocks, equivalent to the pre-shared secret length when multiplied by 16
- [in] `serverHelloRnd`: `ServerHello.random` (concatenated with values already contained in `A71CH`)
- [in] `serverHelloRndLen`: The length of `serverHelloRnd` passed as an argument
- [inout] `masterSecret`: IN: caller passes a buffer of at least 48 byte; OUT: contains the calculated master Secret, TLS 1.2 mandates this to be 48 byte exact

#### Return Value

- `SW_OK`: Successfull execution

U16 **A71\_EcdhPskDeriveMasterSecret** (`SST_Index_t indexKp`, `const U8 *publicKey`, `U16 publicKeyLen`, `SST_Index_t index`, `U8 nBlock`, `const U8 *serverHelloRnd`, `U16 serverHelloRndLen`, `U8 *masterSecret`)

This function calculates the PRF according to TLS1.2 [RFC5246]. The pre-master secret is formed - based upon a pre-shared secret (PSK) stored in the secure module and on an ECDH calculation - according to [RFC5489].

The pre-shared secret is stored in the SYM key store. It can be either 16, 32, 48 or 64 byte long. The Most Significant part of the pre-shared secret resides in the storage location with the lowest index. The subsequent parts reside in the next storage locations. The `nBlock` parameter is equal to the length of the PSK divided by 16.

A PSK cannot be stored wrapped around in the SYM key store.

The PRF creating the `masterSecret` also takes as parameter the concatenation of label ("master\_secret"), `ClientHello.random` and `ServerHello.random`. This function only takes `ServerHello.random` as parameter: `ClientHello.random` has already been set by a call to `A71_CreateClientHelloRandom`, the value of the label (default is "master\_secret") can be overruled by a call to `A71_SetTlsLabel`.

**Pre** This call must be preceded by a call to `A71_CreateClientHelloRandom`, no other `A71CH` API call (implying an APDU exchange between Host and `A71CH`) may be executed in between the invocation of `A71_CreateClientHelloRandom` and `A71_EcdhPskDeriveMasterSecret`

#### Parameters

- [in] `indexKp`: Index of the ECC keypair whose private key is used in the ECDH operation
- [in] `publicKey`: Value of the public key to be used in ECDH operation
- [in] `publicKeyLen`: Length of `publicKey` in byte
- [in] `index`: Index of the SYM key store containing the MSB part of the pre-shared secret
- [in] `nBlock`: Amount of blocks, equivalent to the pre-shared secret length when multiplied by 16
- [in] `serverHelloRnd`: `ServerHello.random` (concatenated with values already contained in `A71CH`)
- [in] `serverHelloRndLen`: The length of `serverHelloRnd` passed as an argument
- [inout] `masterSecret`: IN: caller passes a buffer of at least 48 byte; OUT: contains the calculated master Secret, TLS 1.2 mandates this to be 48 byte exact

#### Return Value

- `SW_OK`: Successfull execution

U16 **A71\_GetHmacSha256** (`SST_Index_t index`, `U8 nBlock`, `const U8 *data`, `U16 dataLen`, `U8 *hmac`, `U16 *hmacLen`)

Calculates the HMAC on `data` using SHA256 as Hash Function according to [RFC2104]. The secret is stored in the SYM key store. It can be either 16, 32, 48 or 64 byte long. The Most Significant part of the secret resides



in the storage location with the lowest index. The subsequent parts reside in the next storage locations. The `nBlock` parameter is equal to the length of the secret divided by 16. A secret with length 64 can only start at Index 0 of the SYM key store; a secret can not be stored wrapped around in the SYM key store.

#### Parameters

- [in] `index`: Index of the SYM key store containing the MSB part of the pre-shared secret
- [in] `nBlock`: Amount of blocks, equivalent to the pre-shared secret length when multiplied by 16
- [in] `data`: Data buffer for which the HMAC-SHA256 must be calculated
- [in] `dataLen`: The length of data passed as argument
- [inout] `hmac`: IN: caller passes a buffer of at least 32 byte; OUT: contains the calculated hmac
- [inout] `hmacLen`: IN: length of the hmac buffer passed; OUT: because SHA256 is used this is 32 byte exact

#### Return Value

- `SW_OK`: Successfull execution

U16 **A71\_HmacSha256Init** (SST\_Index\_t *index*, U8 *nBlock*)  
Initialise multistep HMACSHA256.

#### Parameters

- [in] `index`: Index of the SYM key store containing the MSB part of the pre-shared secret
- [in] `nBlock`: Amount of blocks, equivalent to the pre-shared secret length when multiplied by 16

#### Return Value

- `SW_OK`: Upon successful execution

U16 **A71\_HmacSha256Update** (SST\_Index\_t *index*, U8 *nBlock*, U8 \**data*, U16 *dataLen*)  
Update the data for calculating HMACSHA256 value (in multistep).

#### Parameters

- [in] `index`: Index of the SYM key store containing the MSB part of the pre-shared secret
- [in] `nBlock`: Amount of blocks, equivalent to the pre-shared secret length when multiplied by 16
- [in] `data`: Data buffer for which the HMACSHA256 must be calculated
- [in] `dataLen`: The length of data passed as argument

#### Return Value

- `SW_OK`: Upon successful execution

U16 **A71\_HmacSha256Final** (SST\_Index\_t *index*, U8 *nBlock*, U8 \**hmac*, U16 \**hmacLen*)  
calculating HMACSHA256 value (in multistep).

#### Parameters

- [in] `index`: Index of the SYM key store containing the MSB part of the pre-shared secret
- [in] `nBlock`: Amount of blocks, equivalent to the pre-shared secret length when multiplied by 16
- [inout] `hmac`: IN: caller passes a buffer of at least 32 byte; OUT: contains the calculated HMAC-SHA256
- [inout] `hmacLen`: IN: length of the sha buffer passed; OUT: because HMACSHA256 is used this is 32 byte exact

#### Return Value

- SW\_OK: Upon successful execution

## Secure Storage (SST) API

**Description** Wrap the secure storage functionality of the A71CH.

### Functions

U16 **A71\_SetEccKeyPair** (SST\_Index\_t *index*, const U8 \**publicKey*, U16 *publicKeyLen*, const U8 \**privateKey*, U16 *privateKeyLen*)

Sets an ECC Key pair at storage location *index* with the provided values for public and private key. The private key can optionally be RFC3944 wrapped. Whether wrapping is applied or not is implicit in the length of the private key.

#### Parameters

- [in] *index*: Storage index of the keypair to be created.
- [in] *publicKey*: Pointer to the byte array containing the public key. The public key must be in ANSI X9.62 uncompressed format (including the leading 0x04 byte).
- [in] *publicKeyLen*: Length of the public key (65 byte)
- [in] *privateKey*: Pointer to the byte array containing the private key. The private key may be RFC3394 wrapped using the config key stored at index A71CH\_CFG\_KEY\_IDX\_PRIVATE\_KEYS
- [in] *privateKeyLen*: Length of the private key (either 32 byte for keys in plain format or 40 byte for keys in RFC3944 wrapped format)

#### Return Value

- SW\_OK: Upon successful execution

U16 **A71\_GetPublicKeyEccKeyPair** (SST\_Index\_t *index*, U8 \**publicKey*, U16 \**publicKeyLen*)

Retrieves the ECC Public Key - from a key pair - from the storage location *index* into the provided buffer. The public key retrieved is in ANSI X9.62 uncompressed format (including the leading 0x04 byte).

#### Parameters

- [in] *index*: Storage index of the key pair
- [inout] *publicKey*: IN: buffer to contain public key byte array; OUT: public key
- [inout] *publicKeyLen*: IN: size of provided buffer; OUT: Length of the retrieved public key

#### Return Value

- SW\_OK: Upon successful execution
- ERR\_BUF\_TOO\_SMALL: *publicKey* buffer is too small

U16 **A71\_GetEccKeyPairUsage** (SST\_Index\_t *index*, U8 \**restricted*, U16 \**usedCnt*, U16 \**maxUseCnt*)

Retrieve the usage counter (i.e. how much times the key pair has been used so far to sign) and the maximum usage counter. If the key pair is NOT restricted, usage counter and maximum usage counter will be set to 0 and the restricted parameter will be 0 (otherwise it is 1)

#### Parameters

- [in] *index*: Storage index of the key pair
- [out] *restricted*: 0 when the key pair on *index* is not restricted; 1 if it is restricted. A restricted key pair is a key pair that can only be used to sign a dedicated area in GP storage.

- [out] `usedCnt`: Number of times the key pair on `index` has been used to sign (assuming it is a restricted key pair)
- [out] `maxUseCnt`: Indicates the maximum amount of signing operations associated with the key pair at `index`. In case the value is zero, there is no limit on the amount of signing operations.

U16 **A71\_FreezeEccKeyPair** (SST\_Index\_t *index*)

Freezes an ECC key pair at storage location `index`. This means that the key pair can no longer be erased or its value changed.

**Pre** INJECTION\_LOCKED has not been set

#### Parameters

- [in] `index`: Storage index of the key pair to be frozen.

#### Return Value

- SW\_OK: Upon successful execution

U16 **A71\_FreezeEccKeyPairWithChallenge** (SST\_Index\_t *index*, U8 \**configKey*, U16 *configKeyLen*)

Freezes an ECC key pair at storage location `index`. This means that the key pair can no longer be erased or its value changed. This function must be called instead of `A71_FreezeEccKeyPair` in case `INJECTION_LOCKED` was set

The assumption is the value of the Key Pair configuration key is known on the host. If this does not apply use `A71_FreezeEccKeyPairWithCode` instead.

#### Parameters

- [in] `index`: Storage index of the key pair to be frozen.
- [in] `configKey`: Value of Key Pair configuration key. This value has a high level of confidentiality and may not be available to the Host.
- [in] `configKeyLen`: Length of Key Pair configuration key

#### Return Value

- SW\_OK: Upon successful execution

U16 **A71\_FreezeEccKeyPairWithCode** (SST\_Index\_t *index*, U8 \**code*, U16 *codeLen*)

Freezes an ECC key pair at storage location `index` provided the correct code value is passed as argument. Freezing the key pair means that it can no longer be erased or its value changed. This function must be called instead of `A71_FreezeEccKeyPair` in case `INJECTION_LOCKED` was set.

The assumption is the value of the Key Pair configuration key is not known on the host. If this does not apply use `A71_FreezeEccKeyPairWithChallenge` instead.

The code is calculated as follows:

- Request a challenge from A71CH using `A71_GetUnlockChallenge`.
- Decrypt the challenge in ECB mode using the appropriate configuration key value (the same as stored at index `A71CH_CFG_KEY_IDX_PRIVATE_KEYS`).
- The decrypted value is the value of `code`

#### Parameters

- [in] `index`: Storage index of the key pair to be frozen.
- [in] `code`: Value of unlock code
- [in] `codeLen`: Length of unlock code (must be 16)

### Return Value

- `SW_OK`: Upon successful execution

U16 **A71\_EraseEccKeyPair** (SST\_Index\_t *index*)

Erases an ECC key pair at storage location *index*. This means that the key pair can no longer be used before a new value is set.

**Pre** `INJECTION_LOCKED` has not been set

### Parameters

- [*in*] *index*: Storage index of the key pair to be frozen.

### Return Value

- `SW_OK`: Upon successful execution

U16 **A71\_EraseEccKeyPairWithChallenge** (SST\_Index\_t *index*, U8 \**configKey*, U16 *configKeyLen*)

Erases an ECC key pair at storage location *index*. This means that the key pair can no longer be used before a new value is set. This function must be called instead of `A71_EraseEccKeyPair` in case `INJECTION_LOCKED` was set.

The assumption is the value of the Key Pair configuration key is known on the host. If this does not apply use `A71_EraseEccKeyPairWithCode` instead.

### Parameters

- [*in*] *index*: Storage index of the key pair to be frozen.
- [*in*] *configKey*: Value of Key Pair configuration key. This value has a high level of confidentiality and may not be available to the Host.
- [*in*] *configKeyLen*: Length of Key Pair configuration key

### Return Value

- `SW_OK`: Upon successful execution

U16 **A71\_EraseEccKeyPairWithCode** (SST\_Index\_t *index*, U8 \**code*, U16 *codeLen*)

Erases an ECC key pair at storage location *index*. This means that the key pair can no longer be used before a new value is set. This function must be called instead of `A71_EraseEccKeyPair` in case `INJECTION_LOCKED` was set.

The assumption is the value of the Key Pair configuration key is not known on the host. If this does not apply use `A71_EraseEccKeyPairWithChallenge` instead.

The code is calculated as follows:

- Request a challenge from A71CH using `A71_GetUnlockChallenge`.
- Decrypt the challenge in ECB mode using the appropriate configuration key value (the same as stored at index `A71CH_CFG_KEY_IDX_PRIVATE_KEYS`).
- The decrypted value is the value of *code*

### Parameters

- [*in*] *index*: Storage index of the key pair to be frozen.
- [*in*] *code*: Value of unlock code
- [*in*] *codeLen*: Length of unlock code (must be 16)

### Return Value

- `SW_OK`: Upon successful execution

U16 **A71\_SetEccPublicKey** (SST\_Index\_t *index*, const U8 \**publicKey*, U16 *publicKeyLen*)

Sets an ECC Public Key at storage location *index* with the provided value for public key either in plain ANSI X9.62 uncompressed format or wrapped. Whether RFC3944 wrapping is applied or not is implicit in the length of the public key. In case RFC3944 wrapping is applied the first byte of the public key (the one indicating the public key format) is removed before applying wrapping.

#### Parameters

- [in] *index*: Storage index of the public key to be set.
- [in] *publicKey*: Pointer to the byte array containing the public key. The public key may be RFC3394 wrapped using the config key stored at index A71CH\_CFG\_KEY\_IDX\_PUBLIC\_KEYS
- [in] *publicKeyLen*: Length of the public key (either 65 byte for keys in plain format or 72 byte for keys in RFC3944 wrapped format)

#### Return Value

- SW\_OK: Upon successful execution

U16 **A71\_GetEccPublicKey** (SST\_Index\_t *index*, U8 \**publicKey*, U16 \**publicKeyLen*)

Retrieves the ECC Public Key from the storage location *index* into the provided buffer. The public key is in ANSI X9.62 uncompressed format (including the leading 0x04 byte).

#### Parameters

- [in] *index*: Storage index of the public key to be retrieved.
- [inout] *publicKey*: IN: buffer to contain public key byte array; OUT: public key
- [inout] *publicKeyLen*: IN: size of provided buffer; OUT: Length of the retrieved public key

#### Return Value

- SW\_OK: Upon successful execution
- ERR\_BUF\_TOO\_SMALL: *publicKey* buffer is too small

U16 **A71\_FreezeEccPublicKeyWithChallenge** (SST\_Index\_t *index*, U8 \**configKey*, U16 *configKeyLen*)

Freezes an ECC public key at storage location *index*. This means that the public key can no longer be erased or its value changed. This function must be called instead of A71\_FreezeEccPublicKey in case INJECTION\_LOCKED was set.

The assumption is the value of the Public Key configuration key is known on the host. If this does not apply use A71\_FreezeEccPublicKeyWithCode instead.

#### Parameters

- [in] *index*: Storage index of the public key to be frozen.
- [in] *configKey*: Value of Public Key Pair key. This value has a high level of confidentiality and may not be available to the Host.
- [in] *configKeyLen*: Length of Public Key configuration key

#### Return Value

- SW\_OK: Upon successful execution

U16 **A71\_FreezeEccPublicKeyWithCode** (SST\_Index\_t *index*, U8 \**code*, U16 *codeLen*)

Freezes an ECC public key at storage location *index* provided the correct code value is passed as argument. Freezing the public key means that it can no longer be erased or its value changed. This function must be called instead of A71\_FreezeEccPublicKey in case INJECTION\_LOCKED was set.

The assumption is the value of the Public Key configuration key is not known on the host. If this does not apply use `A71_FreezeEccPublicKeyWithChallenge` instead.

The code is calculated as follows:

- Request a challenge from A71CH using `A71_GetUnlockChallenge`.
- Decrypt the challenge in ECB mode using the appropriate configuration key value (the same as stored at index `A71CH_CFG_KEY_IDX_PUBLIC_KEYS`).
- The decrypted value is the value of `code`

#### Parameters

- [in] `index`: Storage index of the key pair to be frozen.
- [in] `code`: Value of unlock code
- [in] `codeLen`: Length of unlock code (must be 16)

#### Return Value

- `SW_OK`: Upon successful execution

U16 **A71\_FreezeEccPublicKey** (SST\_Index\_t *index*)

Freezes an ECC public key at storage location `index`. This means that the public key can no longer be erased or its value changed.

#### Parameters

- [in] `index`: Storage index of the public key to be frozen.

#### Return Value

- `SW_OK`: Upon successful execution

U16 **A71\_EraseEccPublicKey** (SST\_Index\_t *index*)

Erases an ECC public key at storage location `index`. This means that the public key can no longer be used before a new value is set.

**Pre** `INJECTION_LOCKED` has not been set

#### Parameters

- [in] `index`: Storage index of the public key to be frozen.

#### Return Value

- `SW_OK`: Upon successful execution

U16 **A71\_EraseEccPublicKeyWithChallenge** (SST\_Index\_t *index*, U8 \**configKey*, U16 *configKeyLen*)

Erases an ECC public key at storage location `index`. This means that the public key can no longer be used before a new value is set. This function must be called instead of `A71_EraseEccPublicKey` in case `INJECTION_LOCKED` was set.

The assumption is the value of the Public Key configuration key is known on the host. If this does not apply use `A71_EraseEccPublicKeyWithCode` instead.

#### Parameters

- [in] `index`: Storage index of the public key to be frozen.
- [in] `configKey`: Value of Public Key Pair key. This value has a high level of confidentiality and may not be available to the Host.
- [in] `configKeyLen`: Length of Public Key configuration key

**Return Value**

- `SW_OK`: Upon successful execution

U16 **A71\_EraseEccPublicKeyWithCode** (SST\_Index\_t *index*, U8 \**code*, U16 *codeLen*)

Erases an ECC public key at storage location *index*. This means that the public key can no longer be used before a new value is set. This function must be called instead of `A71_EraseEccPublicKey` in case `INJECTION_LOCKED` was set.

The assumption is the value of the Public Key configuration key is not known on the host. If this does not apply use `A71_EraseEccPublicKeyWithChallenge` instead.

The code is calculated as follows:

- Request a challenge from A71CH using `A71_GetUnlockChallenge`.
- Decrypt the challenge in ECB mode using the appropriate configuration key value (the same as stored at index `A71CH_CFG_KEY_IDX_PUBLIC_KEYS`).
- The decrypted value is the value of *code*

**Parameters**

- [in] *index*: Storage index of the key pair to be frozen.
- [in] *code*: Value of unlock code
- [in] *codeLen*: Length of unlock code (must be 16)

**Return Value**

- `SW_OK`: Upon successful execution

U16 **A71\_SetSymKey** (SST\_Index\_t *index*, **const** U8 \**key*, U16 *keyLen*)

Sets a symmetric key at storage location *index* with the key value. The key locations indexed are the same as the one referenced by `A71_SetRfc3394WrappedAesKey`

**Parameters**

- [in] *index*: Storage index of the symmetric key to be set.
- [in] *key*: Pointer to the byte array containing the symmetric key
- [in] *keyLen*: Length of the symmetric key (must be 16)

**Return Value**

- `SW_OK`: Upon successful execution

U16 **A71\_SetRfc3394WrappedAesKey** (SST\_Index\_t *index*, **const** U8 \**key*, U16 *keyLen*)

Sets an RFC3394 wrapped AES key in secure storage. The key value being set, must be wrapped with the value already stored at *index*. The key locations indexed are the same as the one referenced by `A71_SetSymKey`

**Parameters**

- [in] *index*: index of the key to be set. At the same time the index of the wrapping key.
- [in] *key*: Pointer to the supplied key data.
- [in] *keyLen*: Length of the supplied key data.

**Return Value**

- `SW_OK`: Upon successful execution

U16 **A71\_FreezeSymKey** (SST\_Index\_t *index*)

Freezes a symmetric key at storage location *index*. This means the value of the key at the specified index can no longer be changed.

**Parameters**

- [in] `index`: Storage index of the symmetric key to be frozen.

**Return Value**

- `SW_OK`: Upon successful execution

U16 **A71\_EraseSymKey** (*SST\_Index\_t index*)

Erases the symmetric key at storage location `index`. This means the value of the key at the specified index is cleared. The value must be set anew before the key can be used.

**Parameters**

- [in] `index`: Storage index of the symmetric key to be set.

**Return Value**

- `SW_OK`: Upon successful execution

U16 **A71\_IncrementCounter** (*SST\_Index\_t index*)

Increments the monotonic counter at storage location `index` by one.

**Parameters**

- [in] `index`: Storage index of the counter.

**Return Value**

- `SW_OK`: Upon successful execution

U16 **A71\_SetCounter** (*SST\_Index\_t index*, *U32 value*)

Sets the value of the monotonic counter at storage location ‘`index`’ with the value passed as parameter.

**Parameters**

- [in] `index`: Storage index of the counter.
- [in] `value`: Counter value to be set

**Return Value**

- `SW_OK`: Upon successful execution

U16 **A71\_GetCounter** (*SST\_Index\_t index*, *U32 \*pValue*)

Gets the value of the monotonic counter at storage location ‘`index`’.

**Parameters**

- [in] `index`: Storage index of the counter.
- [out] `pValue`: Counter value retrieved

**Return Value**

- `SW_OK`: Upon successful execution

U16 **A71\_DbgEraseCounter** (*SST\_Index\_t index*)

Sets the value of the monotonic counter at storage location ‘`index`’ to zero.

**Note** Only available when the applet is in Debug Mode.

**Parameters**

- [in] `index`: Storage index of the counter.



**Return Value**

- SW\_OK: Upon successful execution

U16 **A71\_SetGpData** (U16 *dataOffset*, **const** U8 *\*data*, U16 *dataLen*)

Sets a data chunk of General Purpose storage in the security module. Depending on the size of the chunk, this requires one or more APDU exchanges with the security module.

**Pre** The addressed General Purpose storage is not locked.

**Note** In case part of the addressed General Purpose storage is locked, only part of the provided data will have been written most likely leading to an inconsistent data set stored in General Purpose storage.

**Parameters**

- [in] *dataOffset*: Offset for the data in the GP Storage.
- [in] *data*: IN: buffer containing data to write
- [in] *dataLen*: Amount of data to write

**Return Value**

- SW\_OK: Upon successful execution

U16 **A71\_SetGpDataWithLockCheck** (U16 *dataOffset*, **const** U8 *\*data*, U16 *dataLen*)

Sets a data chunk of General Purpose storage in the security module. Depending on the size of the chunk, this requires one or more APDU exchanges with the security module.

**Pre** The addressed General Purpose storage is not locked.

**Note** In case more than one apdu is required, this function first validates that each of the chunks of the addressed General Purpose storage is not locked, and only in that case try to write the provided data using A71\_SetGpData()

**Parameters**

- [in] *dataOffset*: Offset for the data in the GP Storage.
- [in] *data*: IN: buffer containing data to write
- [in] *dataLen*: Amount of data to write

**Return Value**

- SW\_OK: Upon successful execution

U16 **A71\_GetGpData** (U16 *dataOffset*, U8 *\*data*, U16 *dataLen*)

Retrieve a chunk of data from general purpose (GP) storage.

**Parameters**

- [in] *dataOffset*: Offset for the data in the GP Storage.
- [inout] *data*: IN: buffer to contain data; OUT: retrieved data
- [in] *dataLen*: Amount of data to retrieve

**Return Value**

- SW\_OK: Upon successful execution

U16 **A71\_FreezeGpData** (U16 *dataOffset*, U16 *dataLen*)

Mark a chunk in GP storage as frozen (meaning further modification of the GP storage area is disallowed). Both the *dataOffset* and *dataLen* must be aligned on A71CH\_GP\_STORAGE\_GRANULARITY

**Parameters**

- [in] `dataOffset`: Offset for the data in the GP Storage.
- [in] `dataLen`: Amount of data to freeze

#### Return Value

- `SW_OK`: Upon successful execution

U16 **A71\_SetConfigKey** (SST\_Index\_t *index*, const U8 \**key*, U16 *keyLen*)

Sets a config key at storage location *index* with the key value. The key locations indexed are the same as the one referenced by `A71_SetRfc3394WrappedConfigKey`

#### Parameters

- [in] *index*: Storage index of the config key to be set.
- [in] *key*: Pointer to the byte array containing the config key
- [in] *keyLen*: Length of the config key (must be 16)

#### Return Value

- `SW_OK`: Upon successful execution

U16 **A71\_SetRfc3394WrappedConfigKey** (SST\_Index\_t *index*, const U8 \**key*, U16 *keyLen*)

Sets an RFC3394 wrapped config key in secure storage. The key value being set, must be wrapped with the value already stored at the *index*. The key locations indexed are the same as the one referenced by `A71_SetConfigKey`

#### Parameters

- [in] *index*: index of the key to be set. At the same time the index of the wrapping key.
- [in] *key*: Pointer to the supplied key data.
- [in] *keyLen*: Length of the supplied key data.

#### Return Value

- `SW_OK`: Upon successful execution

## sm\_connect.c

**Description** Implementation of basic communication functionality between Host and A71CH. (This file was renamed from `a71ch_com.c` into `sm_connect.c`.)

## Functions

U16 **SM\_RjctConnect** (void \*\**conn\_ctx*, const char \**connectString*, SmCommState\_t \**commState*, U8 \**atr*, U16 \**atrLen*)

U16 **SM\_I2CConnect** (void \*\**conn\_ctx*, SmCommState\_t \**commState*, U8 \**atr*, U16 \**atrLen*, const char \**pConnString*)

U16 **SM\_Connect** (void \**conn\_ctx*, SmCommState\_t \**commState*, U8 \**atr*, U16 \**atrLen*)

Establishes the communication with the Security Module (SM) at the link level and selects the A71CH applet on the SM. The physical communication layer used (e.g. I2C) is determined at compilation time.

#### Parameters

- [inout] *commState*:
- [inout] *atr*:

- [inout] *atrLen*:

#### Return Value

- **SW\_OK**: Upon successful execution

U16 **SM\_Close** (void *\*conn\_ctx*, U8 *mode*)

Closes the communication with the Security Module A new connection can be established by calling **SM\_Connect**

#### Parameters

- [in] *mode*: Specific information that may be required on the link layer

#### Return Value

- **SW\_OK**: Upon successful execution

U16 **SM\_SendAPDU** (U8 *\*cmd*, U16 *cmdLen*, U8 *\*resp*, U16 *\*respLen*)

Sends the command APDU to the Secure Module and retrieves the response APDU. The latter consists of the concatenation of the response data (possibly none) and the status word (2 bytes).

The command APDU and response APDU are not interpreted by the host library.

The command/response APDU sizes must lay within the APDU size limitations

#### Parameters

- [in] *cmd*: command APDU
- [in] *cmdLen*: length (in byte) of *cmd*
- [inout] *resp*: response APDU (response data || response status word)
- [inout] *respLen*: IN: Length of *resp* buffer (*resp*) provided; OUT: effective length of response retrieved.

#### Return Value

- **SW\_OK**: Upon successful execution

### ax\_scp.c

**Description** Set up the SCP03 communication channel.

### Functions

U16 **SCP\_HostLocal\_GetSessionState** (ChannelId\_t *channelId*, Scp03SessionState\_t *\*pSession*)

Copy the session state into *pSession*. Caller must allocate memory of *pSession*.

#### Parameters

- [in] *channelId*: Either **::AX\_HOST\_CHANNEL** or **::AX\_ADMIN\_CHANNEL**. Must be **::AX\_HOST\_CHANNEL** in case of A71CH.
- [inout] *pSession*: IN: pointer to allocated **::Scp03SessionState\_t** structure; OUT: retrieved state

#### Return Value

- **SW\_OK**: Upon successful execution
- **SCP\_UNDEFINED\_CHANNEL\_ID**: In case an undefined **::ChannelId\_t** type was passed as parameter

U16 **SCP\_GetScpSessionState** (Scp03SessionState\_t \*scp03state)

Retrieve the SCP03 session state of the host - secure module channel from the Host Library.

**Return** SW\_OK

**Parameters**

- [inout] scp03state: IN: pointer to allocated structure; OUT: datastructure contains SCP03 session state

void **SCP\_SetScpSessionState** (Scp03SessionState\_t \*scp03state)

Sets SCP03 session state of the host - secure module channel of the Host Library. Can be used in a scenario where e.g. the bootloader has established the SCP03 link between host and secure module and the Host OS must re-establish the communication with the secure module without breaking the SCP03 session.

**Parameters**

- [in] scp03state: IN: SCP03 session state

U16 **SCP\_GP\_ExternalAuthenticate** (ChannelId\_t channelId, U8 \*hostCryptogram)

U16 **SCP\_GP\_InitializeUpdate** (ChannelId\_t channelId, U8 \*hostChallenge, U16 hostChallengeLen, U8 \*keyDivData, U16 \*pKeyDivDataLen, U8 \*keyInfo, U16 \*pKeyInfoLen, U8 \*cardChallenge, U16 \*pCardChallengeLen, U8 \*cardCryptoGram, U16 \*pCardCryptoGramLen, U8 \*seqCounter, U16 \*pSeqCounterLen)

U16 **SCP\_GP\_PutKeys** (U8 keyVersion, U8 \*keyEnc, U8 \*keyMac, U8 \*keyDek, U8 \*currentKeyDek, U16 keyBytes)

Persistently stores the provided SCP03 base key set in the security module.

This method must be called once before the Host - Secure Module SCP channel can be established.

**Return** SW\_OK upon success

**Parameters**

- [in] keyVersion:
- [in] keyEnc: SCP03 channel encryption base key
- [in] keyMac: SCP03 authentication base key
- [in] keyDek: SCP03 data encryption base key
- [in] currentKeyDek: Value of the data encryption base key already stored in secure module, may be NULL in case no key is currently stored.
- [in] keyBytes: Length (in byte) of the keys being set. Typically 16 (corresponding to 128 bits)

U16 **SCP\_Authenticate** (U8 \*keyEnc, U8 \*keyMac, U8 \*keyDek, U16 keyBytes, U8 \*sCounter, U16 \*sCounterLen)

Performs an SCP03 authentication with the SM and - when successful - computes the SCP03 session keys and initializes the current Session state.

**Parameters**

- [in] keyEnc: SCP03 channel encryption base key (aka static key) (16 bytes)
- [in] keyMac: SCP03 authentication base key (aka static key) (16 bytes)
- [in] keyDek: SCP03 data encryption base key (aka static key) (16 bytes)
- [in] keyBytes: Must be 16

- [inout] sCounter: SCP03 sequence counter (3 bytes)
- [inout] sCounterLen:

### scp\_a7x.c

**Description** Conditionally apply SCP03 channel encryption (This file was renamed from `scp.c` into `scp_a7x.c`.)

### a71\_debug.c

**Description** Wrap Debug Mode specific APDU's of A71CH.

### Functions

U16 **A71\_DbgReset** (void)

Resets the Secure Module to the initial state.

#### Return Value

- SW\_OK: Upon successful execution

U16 **A71\_DbgDisableDebug** (void)

Permanently disables the Debug API.

#### Return Value

- SW\_OK: Upon successful execution

U16 **A71\_DbgGetFreePersistentMemory** (S16 \*freeMem)

Reports the available persistent memory in the Security Module.

#### Return Value

- SW\_OK: Upon successful execution

U16 **A71\_DbgGetFreeTransientMemory** (S16 \*freeMem)

Reports the available transient memory in the Security Module.

#### Return Value

- SW\_OK: Upon successful execution

U16 **A71\_DbgReflect** (U8 \*sndBuf, U16 sndBufLen, U8 \*rcvBuf, U16 \*rcvBufLen)

Invokes data reflection APDU (facilitates link testing). No check of data payload returned

#### Return

#### Parameters

- [in] sndBuf:
- [in] sndBufLen:
- [inout] rcvBuf:
- [inout] rcvBufLen:

## 8.4 A71CH Legacy HLSE (Generic) API

### 8.4.1 HLSE API

The API is designed to be generic for Secure Elements that hold cryptographic information and perform cryptographic functions. It isolates an application from the details of the cryptographic device such that it does not have to change to interface to a different type of cryptographic device.

This generic layer intends to abstract both the different APDU specs of applets, and the "file system" details, i.e., how each "object" is stored on the SE. For example, in order to enumerate the certificate objects on the card, the implementation should know where the objects are located, how many there are, what their type is, etc. This abstract layer is important for, e.g. a PKCS#11 layer, and for TLS engines that will have to access objects on the Secure Element.

The HLSE API is written in C allowing maximal portability across different platforms.

Each typedef, enum and function starts with the HLSE (=Host Library Secure Element) prefix.

The various Secure Element entities are referred to as Objects. Every Object (e.g. Key) has a unique handle (HLSE\_OBJECT\_HANDLE) and a set of attributes (HLSE\_ATTRIBUTE) whose values can be retrieved (Get) and Set. An HLSE\_OBJECT\_HANDLE can be obtained in two ways: Either returned by HLSE\_EnumerateObjects() if the Object exists on the Secure Element, or returned by HLSE\_CreateObject() for a new Object. This is an abstraction of the actual way in which the API implements these handles.

An HLSE\_ATTRIBUTE is defined as:

```
typedef struct HLSE_ATTRIBUTE {
 HLSE_ATTRIBUTE_TYPE type;
 void* value;
 U16 valueLen;
} HLSE_ATTRIBUTE;
```

For example, a Private RSA key may have the HLSE\_ATTR\_RSA\_MODULUS and HLSE\_ATTR\_RSA\_PUBLIC\_EXPONENT attributes, and their values can be extracted or set.

Key Generation can be obtained by passing a NULL parameter in the HLSE\_ATTR\_OBJECT\_VALUE attribute. This enables to create the key with a generated random data, or to re-generate an existing key by passing NULL in this attribute's parameter.

A set of functions is responsible for performing cryptographic operations:

- HLSE\_Digest()
- HLSE\_Sign()
- HLSE\_VerifySignature()
- HLSE\_DeriveKey()
- HLSE\_Encrypt()
- HLSE\_Decrypt()

The cryptographic algorithm is controlled by a HLSE\_MECHANISM\_INFO, defined as:

```
typedef struct HLSE_MECHANISM_INFO {
 HLSE_MECHANISM_TYPE mechanism;
 void* pParameter;
 U16 ulParameterLen;
} HLSE_MECHANISM_INFO;
```

A list of the supported mechanisms, either by the library or by a specific key, can be obtained by calling `HLSE_GetSupportedMechanisms()` or `HLSE_GetSupportedMechanismsForObject()`, respectively.

The HLSE API is made up of four parts:

- Operations on Objects (*HLSEObjects.h*)
- Cryptographic operations (*HLSECrypto.h*)
- Secure Element Communication and Secure Channel management functions (*HLSEComm.h*)
- Debug Mode variant and miscellaneous functionality (*HLSEMisc.h*)

An additional file `.../hostLib/inc/HLSEAPI.h` serves as an entry point to the full API. The implementation of the API dealing with A71CH specific functionality is in `.../hostLib/api/src/A71HLSEWrapper.c`.

## 8.4.2 Logical objects

The HLSE API allows to create logical objects in the GP Storage. They can be of `HLSE_CERTIFICATE` or `HLSE_DATA` object type. The abstraction for various objects that reside in the GP Storage area is achieved by maintaining a lookup table (mapping) at the end of the GP Storage area to hold information about the logical objects that exist in the GP Storage. The structure of the table is as follows:

|                                                        |                         |          |
|--------------------------------------------------------|-------------------------|----------|
| Notes:                                                 |                         |          |
| X+1 is the address of the last byte of the GP Storage. |                         |          |
| N is the object number from 1 to N                     |                         |          |
| Address                                                | Value                   |          |
| -----                                                  | -----                   |          |
| X-N*6+0                                                | N'th Object Class       | - 1 byte |
| X-N*6+1                                                | N'th Object Index       | - 1 byte |
| X-N*6+2                                                | N'th Object Length MSB  | - 1 byte |
| X-N*6+3                                                | N'th Object Length LSB  | - 1 byte |
| X-N*6+4                                                | N'th Object Offset MSB  | - 1 byte |
| X-N*6+5                                                | N'th Object Offset LSB  | - 1 byte |
|                                                        | ?                       |          |
| X-1*6+0                                                | First Object Class      | - 1 byte |
| X-1*6+1                                                | First Object Index      | - 1 byte |
| X-1*6+2                                                | First Object Length MSB | - 1 byte |
| X-1*6+3                                                | First Object Length LSB | - 1 byte |
| X-1*6+4                                                | First Object Offset MSB | - 1 byte |
| X-1*6+5                                                | First Object Offset LSB | - 1 byte |
|                                                        |                         |          |
| X                                                      | Update Counter          | - 1 byte |
| X+1                                                    | Number of table entries | - 1 byte |
| End of GP Storage                                      |                         |          |

The table will be written so that the ‘Number of table entries’ byte is the last byte of the GP Data (to allow the map to grow dynamically as long as there is enough free space), preceded by one byte of the Update Counter and then preceded by 6-tuples of entries.

The Class byte is equivalent to the object type using a single byte (0x09 for Certificate, 0x0A for Data).

The order of the 6-tuple entries is not important, as each object is identified by its Class and Index. In cases where the length of an object is not known at the time the lookup table entry is created, the MSBit (0x8000) can be set in the length as an indicator that the data is in TLV format and that the actual length must be obtained by reading the first bytes of the object’s data.

For objects of type ‘Certificate’ the provisioned ‘Object Length’ value must be one of the following:

- The reserved object storage length (allowing for a possible increase in size of the certificate or for die-individual variance of the certificate size).
- The actual certificate length
- In the exceptional case neither a reserved certificate object storage length nor the effective certificate length can be determined one can use the value '0x8000' to indicate the 'Object Size' is unknown at the time of provisioning.

When reading a certificate from GP storage with the HLSE API, the size of the certificate is always determined by the length value of the certificate's initial TL(V) header.

The host library reads the total number of entries in the table from the last byte of the GP Data, followed by parsing/reading the 6-tuple entries. Up to 254 (0xFE) objects are assumed. A value of 0xFF in the number of entries indicates that the table is absent (uninitialized) or invalid.

Class and Index value of 0xFF indicates an invalid entry (i.e. of a deleted object).

The Update Counter is initially set to 0 and it is incremented on each table update. This serves as an indication to a GP Storage's change when there is more than one application updating the SE concurrently.

The value of 'object offset' must be a multiple of 32.

It is not allowed to create a data object of size 0.

## **Object creation**

The library supports a dynamic number of objects in the GP Storage, according to the memory availability.

Creating an object through HLSE\_CreateObject() requires the following attributes to be passed:

1. HLSE\_ATTR\_OBJECT\_TYPE ? currently HLSE\_CERTIFICATE or HLSE\_DATA;
2. HLSE\_ATTR\_OBJECT\_INDEX ? will be the Tag of the object, 1 byte;
3. HLSE\_ATTR\_OBJECT\_VALUE ? the object's value.

An additional attribute that can only be passed in Create is HLSE\_ATTR\_READ\_ONLY. Setting this value to 1 will lock ('freeze') the memory associated with the object (once it has been created) so it cannot be modified. The HLSE\_ATTR\_READ\_ONLY attribute is not explicitly stored in the GP Storage lookup table.

Note that this attribute cannot be set after object creation. If not passed, it has a default value of 0 (can be modified).

Creation fails if there is not enough continuous unlocked space for the new object's value.

## **Value Update**

It is possible to change the object's value by calling HLSE\_SetObjectAttribute() with HLSE\_ATTR\_OBJECT\_VALUE. If the object needs to be enlarged, it is only permitted if enough memory is available for the object to grow, case as follows:

1. Within the same GP Storage's chunk size (32 bytes), so that the same amount of storage chunks will be used - For example, if the size was originally 21 bytes then the object occupies 1 chunk, and it is possible to enlarge it up to 32 bytes.
2. Up to the offset of the next allocated object in the GP memory.

If a larger size is required, the object must first be erased and then re-created (assuming a sufficiently large continuous unlocked space is available in GP memory).



## Direct Access Value Update

It is possible to change a sub section of a Data object's value by calling `HLSE_SetObjectAttribute()` with `HLSE_ATTR_DIRECT_ACCESS_OBJECT_VALUE`, where the value should point to a `HLSE_DIRECT_ACCESS_ATTRIBUTE_VALUE` structure that passes the offset, number of bytes to read and the buffer. The update is only permitted within the object's GP Storage's chunk boundary.

## Erasing an object

To erase an object first fetch its handle with `HLSE_EnumerateObjects()` and call `HLSE_EraseObject()`. Erasing an object only invalidates its lookup table entry, it does not erase its value contents in the GP Storage, due to performance reasons.

## Interoperability of Object storage and locked chunks

The following defines the behavior of the HLSE API when updating (full/partial) or erasing partially locked objects stored in GP memory:

1. When updating an object (by definition this concerns the complete object) the HLSE API first checks that no chunk of the object is locked before updating the object.
2. When doing a partial update of an object the HLSE API does not check whether the affected memory chunks(s) are locked or unlocked. The partial update will fail or succeed accordingly. Consequently one must only issue a partial update of an object for chunks that are unlocked.
3. When erasing an object, the `HLSE_EnumerateObjects()` API checks whether the first chunk of the object is locked. If the first chunk is not locked the entry corresponding to the object is removed from the GP lookup table. If the first chunk is locked, the object is considered 'read-only' and the object is not removed from the GP lookup table. As explained above, erasing an object does not erase the value associated with the object.
4. Locking the GP storage chunks containing the lookup table (even a partial lock) will make it impossible to remove, add or update objects.

## Notes

1. If the applet is Trust Provisioned prior to being shipped to the user, with e.g. one or more certificate(s), then the lookup table is expected to be in the GP Data.
2. If the lookup table is missing (invalid value), then it is automatically created by the host library upon the first call to `CreateObject` of such an object.
3. When reading a certificate, the response omits any trailing padding at the end of the certificate. The size of the certificate is determined by the length value of the certificates initial TL(V) header.
4. When updating an Object - the length in the GP table will be kept as the maximum size of the existing and new the object. As a consequence, it's not possible to shrink the size of an object by updating it.
5. Don't use direct A71CH API access in combination with the HLSE Object API as one can damage the lookup table or the value of stored objects.

### 8.4.3 API details

#### HLSEObjects.h

**Description** Host Lib wrapper API: Object Operations

#### Functions

HLSE\_RET\_CODE **HLSE\_EnumerateObjects** (HLSE\_OBJECT\_TYPE *objectType*,  
HLSE\_OBJECT\_HANDLE \**objectHandles*, U16  
\**objectHandlesLen*)

Enumerates all the Objects that currently exist on the Secure Element and have *objectType* type. A list of object handles is returned in *objectHandles*.

In order to enumerate all the Objects, set HLSE\_ANY\_TYPE in *objectType*.

Each object has a unique HLSE\_OBJECT\_HANDLE value - this value depends on the library implementation.

If *objectHandles* is NULL, then all that the function does is return (in \**objectHandlesLen*) a number of HLSE\_OBJECT\_HANDLE which would suffice to hold the returned list. HLSE\_SW\_OK is returned by the function.

If *objectHandles* is not NULL, then \**objectHandlesLen* must contain the number of handles in the buffer *objectHandles*. If that buffer is large enough to hold number of handles to be returned, then the handles are copied to *objectHandles*, and HLSE\_SW\_OK is returned by the function. If the buffer is not large enough, then HLSE\_ERR\_BUF\_TOO\_SMALL is returned. In either case, \**objectHandlesLen* is set to hold the exact number of handles to be returned.

#### Parameters

- [in] *objectType*: The type of the Objects to be enumerated
- [inout] *objectHandles*: IN: caller passes a buffer of at least \**objectHandlesLen*; OUT: contains the handles of the objects
- [inout] *objectHandlesLen*: IN: number of handles in *objectHandles*. OUT: set to hold the exact number of handles in *objectHandles*.

#### Return Value

- HLSE\_SW\_OK: Successfull execution
- HLSE\_ERR\_BUF\_TOO\_SMALL: Buffer is too small to return the handles
- HLSE\_ERR\_API\_ERROR: Invalid function arguments

HLSE\_RET\_CODE **HLSE\_CreateObject** (HLSE\_ATTRIBUTE \**attributes*, U16 *attributesNum*,  
HLSE\_OBJECT\_HANDLE \**hObject*)

Creates or Generates an Object on the Secure Element, and returns a handle to it.

If the object already exists, it depends on the Secure Element behavior whether this function succeeds (e.g. set a new value) or fail with an error.

*attributes* is an array of attributes that the object should be created with. Some of the attributes may be mandatory, such as HLSE\_ATTR\_OBJECT\_TYPE and HLSE\_ATTR\_OBJECT\_INDEX (the id of the object), and some are optional.

In case there is a conflict in the attribute list (e.g. 2 differnt object types) it is the responsibility of the library to detect it and return an error.

#### Parameters

- [in] `attributes`: The attributes to be used in creating the Object
- [in] `attributesNum`: The number of attributes in `attributes`
- [inout] `hObject`: IN: A pointer to a handle (must not be NULL); OUT: The handle of the created Object

#### Return Value

- `HLSE_SW_OK`: Successfull execution
- `HLSE_ERR_API_ERROR`: Invalid function arguments

`HLSE_RET_CODE` **HLSE\_EraseObject** (`HLSE_OBJECT_HANDLE` *hObject*)

Erases an object from the Secure Element.

This means the object with the specified handle can no longer be used.

#### Parameters

- [in] `hObject`: The handle of the Object to be erased

#### Return Value

- `HLSE_SW_OK`: Successfull execution

`HLSE_RET_CODE` **HLSE\_SetObjectAttribute** (`HLSE_OBJECT_HANDLE` *hObject*,  
`HLSE_ATTRIBUTE *attribute`)

Sets the requested Attribute of the Object.

The parameter `attribute` may convey additional information (e.g. a key value), in addition to the attribute's type.

#### Parameters

- [in] `hObject`: The handle of the Object that its attribute should be set
- [in] `attribute`: The attribute to be Set

#### Return Value

- `HLSE_SW_OK`: Successfull execution
- `HLSE_ERR_API_ERROR`: Invalid function arguments

`HLSE_RET_CODE` **HLSE\_GetObjectAttribute** (`HLSE_OBJECT_HANDLE` *hObject*,  
`HLSE_ATTRIBUTE *attribute`)

Obtains the value of the Object's requested Attribute.

The parameter `attribute` specifies the Type of the attribute to be returned, and the data is returned in the attribute's value and valueLen members.

If `attribute->value` is NULL, then all that the function does is return (in `*attribute->valueLen`) a number of bytes which would suffice to hold the value to be returned. `HLSE_SW_OK` is returned by the function.

If `attribute->value` is not NULL, then `*attribute->valueLen` must contain the number of bytes in the buffer `attribute->value`. If that buffer is large enough to hold the value be returned, then the data is copied to `attribute->value`, and `HLSE_SW_OK` is returned by the function. If the buffer is not large enough, then `HLSE_ERR_BUF_TOO_SMALL` is returned. In either case, `*attribute->valueLen` is set to hold the exact number of bytes to be returned.

#### Parameters

- [in] `hObject`: The handle of the Object that its attribute's value should be obtained

- [inout] attribute: The attribute to be obtained

#### Return Value

- HLSE\_SW\_OK: Successfull execution
- HLSE\_ERR\_BUF\_TOO\_SMALL: attribute->value is too small to return the data
- HLSE\_ERR\_API\_ERROR: Invalid function arguments

HLSE\_RET\_CODE **Debug\_ForceReadGPDataTable** (void)

Debug Utility

Force Read of GPDataTable from gp storage even if already in global memory variable

*NOTE!! : To be used only for internal testing and Debugging*

currently used to test the GP Table manipulation

#### Return Value

- HLSE\_SW\_OK: Successfull execution

## HLSECrypto.h

**Description** Host Lib wrapper API: Cryptographic functions

### Functions

HLSE\_RET\_CODE **HLSE\_GetSupportedMechanisms** (HLSE\_MECHANISM\_TYPE \*mechanisms,  
U16 \*mechanismNum)

Enumerates all the Cryptographic Mechanisms that are supported by the library. A list of mechanisms is returned in mechanisms.

If mechanisms is NULL, then all that the function does is return (in \*mechanismNum) the number of HLSE\_MECHANISM\_TYPE which would suffice to hold the returned list. HLSE\_SW\_OK is returned by the function.

If mechanisms is not NULL, then \*mechanismNum must contain the number of mechanisms in the buffer mechanisms. If that buffer is large enough to hold number of mechanisms to be returned, then the mechanisms are copied to mechanisms, and HLSE\_SW\_OK is returned by the function. If the buffer is not large enough, then HLSE\_ERR\_BUF\_TOO\_SMALL is returned. In either case, \*mechanismNum is set to hold the exact number of mechanisms to be returned.

#### Parameters

- [inout] mechanisms: IN: caller passes a buffer of at least \*mechanismNum; OUT: contains the mechanisms supported
- [inout] mechanismNum: IN: number of mechanisms in mechanisms; OUT: set to hold the exact number of mechanisms

#### Return Value

- HLSE\_SW\_OK: Successfull execution
- HLSE\_ERR\_BUF\_TOO\_SMALL: Buffer is too small to return the mechanisms
- HLSE\_ERR\_API\_ERROR: Invalid function arguments

HLSE\_RET\_CODE **HLSE\_GetSupportedMechanismsForObject** (HLSE\_OBJECT\_HANDLE  
                                                           *hObject*,  
                                                           HLSE\_MECHANISM\_TYPE  
                                                           *\*mechanism*,      U16      *\*mecha-*  
                                                           *nismLen*)

Enumerates all the Cryptographic Mechanisms that are supported by the Object. A list of mechanisms is returned in *mechanisms*.

If *mechanism* is NULL, then all that the function does is return (in *\*mechanismLen*) the number of HLSE\_MECHANISM\_TYPE which would suffice to hold the returned list. HLSE\_SW\_OK is returned by the function.

If *mechanism* is not NULL, then *\*mechanismLen* must contain the number of mechanisms in the buffer *mechanisms*. If that buffer is large enough to hold number of mechanisms to be returned, then the mechanisms are copied to *mechanisms*, and HLSE\_SW\_OK is returned by the function. If the buffer is not large enough, then HLSE\_ERR\_BUF\_TOO\_SMALL is returned. In either case, *\*mechanismLen* is set to hold the exact number of mechanisms to be returned.

#### Parameters

- [in] *hObject*: The handle of the Object that the Mechanisms it supports should be returned
- [inout] *mechanism*: IN: caller passes a buffer of at least *\*mechanismNum*; OUT: contains the mechanisms supported
- [inout] *mechanismLen*: IN: number of mechanisms in *mechanisms*. OUT: set to hold the exact number of mechanisms

#### Return Value

- HLSE\_SW\_OK: Successful execution
- HLSE\_ERR\_BUF\_TOO\_SMALL: Buffer is too small to return the mechanisms
- HLSE\_ERR\_API\_ERROR: Invalid function arguments

HLSE\_RET\_CODE **HLSE\_Digest** (HLSE\_MECHANISM\_INFO *\*pMechanismType*, U8 *\*inData*, U16 *in-*  
                                                           *DataLen*, U8 *\*outDigest*, U16 *\*outDigestLen*)

Calculates the Digest (e.g. Sha256) value of the data provided as input.

The Cryptographic Mechanism to be used is passed in the *type* member of the *pMechanismType* parameter.

If additional information is required by the specific digest mechanism, it will be conveyed in *pMechanismType->pParameter*.

If *outDigest* is NULL, then all that the function does is return (in *\*outDigestLen*) a number of bytes which would suffice to hold the digest value. HLSE\_SW\_OK is returned by the function.

If *outDigest* is not NULL, then *\*outDigestLen* must contain the number of bytes in the buffer *outDigest*. If that buffer is large enough to hold the digest value to be returned, then the data is copied to *outDigest*, and HLSE\_SW\_OK is returned by the function. If the buffer is not large enough, then HLSE\_ERR\_BUF\_TOO\_SMALL is returned. In either case, *\*outDigestLen* is set to hold the exact number of bytes to be returned.

#### Parameters

- [in] *pMechanismType*: The Digest Cryptographic Mechanism to be used
- [in] *inData*: Data buffer for which the digest must be calculated
- [in] *inDataLen*: The length of data passed as argument

- [inout] outDigest: IN: caller passes a buffer to hold the digest value; OUT: contains the calculated digest
- [inout] outDigestLen: IN: length of the outDigest buffer passed; OUT: the number of bytes returned in outDigest

#### Return Value

- HLSE\_SW\_OK: Upon successful execution
- HLSE\_ERR\_BUF\_TOO\_SMALL: outDigest is too small to return the digest
- HLSE\_ERR\_API\_ERROR: Invalid function arguments

HLSE\_RET\_CODE **HLSE\_Sign** (HLSE\_MECHANISM\_INFO *\*pMechanismType*,  
 HLSE\_OBJECT\_HANDLE *hObject*, U8 *\*inData*, U16 *inDataLen*,  
 U8 *\*outSignature*, U16 *\*outSignatureLen*)

Signs the data provided using the Object key and the requested mechanism.

The Cryptographic Mechanism to be used is passed in the type member of the pMechanismType parameter. A handle to the key to sign the data with is provided by hObject.

If additional information is required by the specific signing mechanism, it will be conveyed in pMechanismType->pParameter.

If outSignature is NULL, then all that the function does is return (in \*outSignatureLen) a number of bytes which would suffice to hold the signature. HLSE\_SW\_OK is returned by the function.

If outSignature is not NULL, then \*outSignatureLen must contain the number of bytes in the buffer outSignature. If that buffer is large enough to hold the signature to be returned, then the data is copied to outSignature, and HLSE\_SW\_OK is returned by the function. If the buffer is not large enough, then HLSE\_ERR\_BUF\_TOO\_SMALL is returned. In either case, \*outSignatureLen is set to hold the exact number of bytes to be returned.

#### Parameters

- [in] pMechanismType: The signing Cryptographic Mechanism to be used
- [in] hObject: The handle of the Object key to sign with
- [in] inData: Data buffer for that should be signed (e.g. a digest)
- [in] inDataLen: The length of data passed as argument
- [inout] outSignature: IN: caller passes a buffer to hold the signature; OUT: contains the calculated signature
- [inout] outSignatureLen: IN: length of the outSignature buffer passed; OUT: the number of bytes returned in outSignature

#### Return Value

- HLSE\_SW\_OK: Upon successful execution
- HLSE\_ERR\_BUF\_TOO\_SMALL: outSignature is too small to return the signature
- HLSE\_ERR\_API\_ERROR: Invalid function arguments

HLSE\_RET\_CODE **HLSE\_VerifySignature** (HLSE\_MECHANISM\_INFO *\*pMechanismType*,  
 HLSE\_OBJECT\_HANDLE *hObject*, U8 *\*inData*,  
 U16 *inDataLen*, U8 *\*inSignature*, U16 *inSignatureLen*)

Verifies whether inSignature is the signature of inData using the public key object referenced by hObject as the verifying public key.

The Cryptographic Mechanism to be used is passed in the type member of the pMechanismType parameter.

**Parameters**

- [in] *pMechanismType*: The signing Cryptographic Mechanism that was used
- [in] *hObject*: The handle of the Object public key to verify with
- [in] *inData*: The data that was signed (e.g. a digest)
- [in] *inDataLen*: The length of data passed as argument
- [in] *inSignature*: Pointer to the provided signature.
- [in] *inSignatureLen*: Length of the provided signature (*pSignature*)

**Return Value**

- *HLSE\_SW\_OK*: Upon successful execution
- *HLSE\_ERR\_GENERAL\_ERROR*: if the verification fails

*HLSE\_RET\_CODE* **HLSE\_VerifySignatureWithExternalKey** (*HLSE\_MECHANISM\_INFO* *\*pMechanismType*, U8 *\*inExtKey*, U16 *inExtKeyLen*, U8 *\*inData*, U16 *inDataLen*, U8 *\*inSignature*, U16 *inSignatureLen*)

Verifies whether *inSignature* is the signature of *inData* using an external public key object as the verifying public key.

The Cryptographic Mechanism to be used is passed in the *type* member of the *pMechanismType* parameter.

**Parameters**

- [in] *pMechanismType*: The signing Cryptographic Mechanism that was used
- [in] *inExtKey*: The value of the external public key to verify with
- [in] *inExtKeyLen*: The length in bytes of the external key
- [in] *inData*: The data that was signed (e.g. a digest)
- [in] *inDataLen*: The length of data passed as argument
- [in] *inSignature*: Pointer to the provided signature.
- [in] *inSignatureLen*: Length of the provided signature (*pSignature*)

**Return Value**

- *HLSE\_SW\_OK*: Upon successful execution
- *HLSE\_ERR\_GENERAL\_ERROR*: if the verification fails

*HLSE\_RET\_CODE* **HLSE\_DeriveKey** (*HLSE\_MECHANISM\_INFO* *\*pMechanismType*, *HLSE\_OBJECT\_HANDLE* *hObject*, U8 *\*outDerivedKey*, U16 *\*outDerivedKeyLen*)

Derives the key referenced by the *hObject* handle using the requested mechanism and return the derived key in *outDerivedKey*.

The Cryptographic Mechanism to be used is passed in the *type* member of the *pMechanismType* parameter.

If additional information is required by the specific signing mechanism, it will be conveyed in *pMechanismType->pParameter*.

If *outDerivedKey* is NULL, then all that the function does is return (in *\*outDerivedKeyLen*) a number of bytes which would suffice to hold the derived key. *HLSE\_SW\_OK* is returned by the function.

If `outDerivedKey` is not NULL, then `*outDerivedKeyLen` must contain the number of bytes in the buffer `outDerivedKey`. If that buffer is large enough to hold the derived key, then the data is copied to `outDerivedKey`, and `HLSE_SW_OK` is returned by the function. If the buffer is not large enough, then `HLSE_ERR_BUF_TOO_SMALL` is returned. In either case, `*outDerivedKeyLen` is set to hold the exact number of bytes of the derived key.

#### Parameters

- [in] `pMechanismType`: The signing Cryptographic Mechanism to be used
- [in] `hObject`: The handle of the Object key to be derived
- [inout] `outDerivedKey`: IN: caller passes a buffer to hold the derived key; OUT: contains the derived key
- [inout] `outDerivedKeyLen`: IN: length of the `outDerivedKey` buffer passed; OUT: the number of bytes returned in `outDerivedKey`

#### Return Value

- `HLSE_SW_OK`: Upon successful execution
- `HLSE_ERR_BUF_TOO_SMALL`: `outDerivedKey` is too small to return the derived key
- `HLSE_ERR_API_ERROR`: Invalid function arguments

`HLSE_RET_CODE HLSE_Encrypt` (`HLSE_MECHANISM_INFO` *\*pMechanismType*,  
`HLSE_OBJECT_HANDLE` *hObject*, `U8` *\*inData*, `U16` *inDataLen*, `U8`  
*\*outData*, `U16` *\*outDataLen*)

Encrypts the data provided using the Object key and the requested mechanism.

The Cryptographic Mechanism to be used is passed in the `type` member of the `pMechanismType` parameter. A handle to the key to encrypt the data with is provided by `hObject`.

If additional information is required by the specific encryption mechanism, it will be conveyed in `pMechanismType->pParameter`.

If `outData` is NULL, then all that the function does is return (in `*outDataLen`) a number of bytes which would suffice to hold the return value. `HLSE_SW_OK` is returned by the function.

If `outData` is not NULL, then `*outDataLen` must contain the number of bytes in the buffer `outData`. If that buffer is large enough to hold the data to be returned, then the data is copied to `outData`, and `HLSE_SW_OK` is returned by the function. If the buffer is not large enough, then `HLSE_ERR_BUF_TOO_SMALL` is returned. In either case, `*outDataLen` is set to hold the exact number of bytes to be returned.

#### Parameters

- [in] `pMechanismType`: The encryption Cryptographic Mechanism to be used
- [in] `hObject`: The handle of the Object key to encrypt with
- [in] `inData`: Data buffer for that should be encrypted
- [in] `inDataLen`: The length of data passed as argument
- [inout] `outData`: IN: caller passes a buffer to hold the data to be returned; OUT: contains the encrypted data
- [inout] `outDataLen`: IN: length of the `outData` buffer passed; OUT: the number of bytes returned in `outData`

#### Return Value

- `HLSE_SW_OK`: Upon successful execution



- HLSE\_ERR\_BUF\_TOO\_SMALL: outData is too small to return the data
- HLSE\_ERR\_API\_ERROR: Invalid function arguments

HLSE\_RET\_CODE **HLSE\_Decrypt** (HLSE\_MECHANISM\_INFO *\*pMechanismType*,  
 HLSE\_OBJECT\_HANDLE *hObject*, U8 *\*inData*, U16 *inDataLen*, U8  
*\*outData*, U16 *\*outDataLen*)

Decrypts the data provided using the Object key and the requested mechanism.

The Cryptographic Mechanism to be used is passed in the `type` member of the `pMechanismType` parameter. A handle to the key to decrypt the data with is provided by `hObject`.

If additional information is required by the specific decryption mechanism, it will be conveyed in `pMechanismType->pParameter`.

If `outData` is NULL, then all that the function does is return (in `*outDataLen`) a number of bytes which would suffice to hold the return value. HLSE\_SW\_OK is returned by the function.

If `outData` is not NULL, then `*outDataLen` must contain the number of bytes in the buffer `outData`. If that buffer is large enough to hold the data to be returned, then the data is copied to `outData`, and HLSE\_SW\_OK is returned by the function. If the buffer is not large enough, then HLSE\_ERR\_BUF\_TOO\_SMALL is returned. In either case, `*outDataLen` is set to hold the exact number of bytes to be returned.

#### Parameters

- [in] `pMechanismType`: The decryption Cryptographic Mechanism to be used
- [in] `hObject`: The handle of the Object key to decrypt with
- [in] `inData`: Data buffer for that should be decrypted
- [in] `inDataLen`: The length of data passed as argument
- [inout] `outData`: IN: caller passes a buffer to hold the data to be returned; OUT: contains the decrypted data
- [inout] `outDataLen`: IN: length of the `outData` buffer passed; OUT: the number of bytes returned in `outData`

#### Return Value

- HLSE\_SW\_OK: Upon successful execution
- HLSE\_ERR\_BUF\_TOO\_SMALL: outData is too small to return the data
- HLSE\_ERR\_API\_ERROR: Invalid function arguments

## HLSEComm.h

**Description** Host Lib wrapper API: Communication and Secure Channel functions

## Functions

HLSE\_RET\_CODE **HLSE\_Connect** (HLSE\_CONNECTION\_PARAMS *\*params*,  
HLSE\_COMMUNICATION\_STATE *\*commState*)

Establishes the communication with the Secure Element according to the requested type. Additional parameters required for establishing the communication are passed in *params*.

The physical communication layer used (e.g. SCI2C) is determined at compilation time.

### Parameters

- [inout] *params*: Additional parameters for opening the communication
- [inout] *commState*: Points to a HLSE\_COMMUNICATION\_STATE which returns the communication state (e.g. ATR)

### Return Value

- HLSE\_SW\_OK: Upon successful execution
- HLSE\_ERR\_API\_ERROR: Invalid function arguments

HLSE\_RET\_CODE **HLSE\_CloseConnection** (HLSE\_CLOSE\_CONNECTION\_MODE *mode*)

Closes the communication with the Secure Element.

### Parameters

- [in] *mode*: Specific information that may be required on the link layer

### Return Value

- HLSE\_SW\_OK: Upon successful execution
- HLSE\_ERR\_API\_ERROR: Invalid function arguments

HLSE\_RET\_CODE **HLSE\_ResumeConnection** (HLSE\_COMMUNICATION\_STATE *\*commState*,  
HLSE\_SECURE\_CHANNEL\_SESSION\_STATE *\*smState*)

Resumes the communication with the Secure Element including the secure channel from the previously retrieved communication state and secure channel session state.

### Parameters

- [in] *commState*: communication state
- [in] *smState*: secure channel session state

### Return Value

- HLSE\_SW\_OK: Upon successful execution
- HLSE\_ERR\_API\_ERROR: Invalid function arguments

HLSE\_RET\_CODE **HLSE\_SendAPDU** (U8 *\*cmd*, U16 *cmdLen*, U8 *\*resp*, U16 *\*respLen*)

Sends the command APDU to the Secure Element and retrieves the response APDU. The latter consists of the concatenation of the response data (possibly none) and the status word (2 bytes).

The command APDU and response APDU are not interpreted by the host library.

The command/response APDU sizes must lay within the APDU size limitations

### Parameters

- [in] *cmd*: command APDU

- [in] cmdLen: length (in byte) of cmd
- [inout] resp: response APDU (response data || response status word)
- [inout] respLen: IN: Length of resp buffer (resp) provided; OUT: effective length of response retrieved.

#### Return Value

- HLSE\_SW\_OK: Upon successful execution
- HLSE\_ERR\_API\_ERROR: Invalid function arguments

HLSE\_RET\_CODE **HLSE\_SCP\_Subscribe** (HLSE\_SCP\_SignalFunction *callback*, void *\*context*)  
 Subscribe a HLSE\_SCP\_SignalFunction function to receive messages from the Secure Channel.

#### Parameters

- [in] callback: The function
- [in] context: Optional context information that the function is subscribed with and called with

#### Return Value

- HLSE\_SW\_OK: Upon successful execution
- HLSE\_ERR\_API\_ERROR: Invalid function arguments

HLSE\_RET\_CODE **HLSE\_SMChannelAuthenticate** (HLSE\_SECURE\_CHANNEL\_ESTABLISH\_PARAMS  
                                                   *\*params*, HLSE\_SECURE\_CHANNEL\_STATE  
                                                   *\*channelState*)

Establishes a Secure Channel with the Secure Element, and when successful initializes the current Session Channel state.

#### Parameters

- [in] params: Data required to establish the Secure Channel
- [inout] channelState: Returns the Secure Channel state

#### Return Value

- HLSE\_SW\_OK: Upon successful execution
- HLSE\_ERR\_API\_ERROR: Invalid function arguments

HLSE\_RET\_CODE **HLSE\_SMChannelGetScpSessionState** (HLSE\_SECURE\_CHANNEL\_SESSION\_STATE  
                                                           *\*channelSessionState*)

Retrieve the Secure Channel Session state from the Host Library.

#### Parameters

- [inout] channelSessionState: IN: pointer to allocated structure; OUT: contains the session state

#### Return Value

- HLSE\_SW\_OK: Upon successful execution
- HLSE\_ERR\_API\_ERROR: Invalid function arguments

HLSE\_RET\_CODE **HLSE\_SMChannelSetScpSessionState** (HLSE\_SECURE\_CHANNEL\_SESSION\_STATE  
                                                           *\*channelSessionState*)

Sets the Secure Channel Session state of the Host Library. Can be used in a scenario where e.g. the bootloader has established the Secure Channel link between host and secure element and the Host OS must re-establish the communication with the secure element without breaking the session.

### Parameters

- [in] `channelSessionState`: Contains the session state information

### Return Value

- `HLSE_SW_OK`: Upon successful execution
- `HLSE_ERR_API_ERROR`: Invalid function arguments

## HLSEMisc.h

**Description** Host Lib wrapper API: Miscellaneous functions

### Functions

`HLSE_RET_CODE HLSE_DisablePlainInjectionMode` (void)  
Permanently disables the Plain Injection mode

#### Return Value

- `HLSE_SW_OK`: Upon successful execution

`HLSE_RET_CODE HLSE_ResetContents` (void)  
Clears all user data.

#### Return Value

- `HLSE_SW_OK`: Upon successful execution

`HLSE_RET_CODE HLSE_DbgDisableDebug` (void)  
Permanently disables the Debug API.

#### Return Value

- `HLSE_SW_OK`: Upon successful execution

`HLSE_RET_CODE HLSE_DbgReflect` (U8 \**inData*, U16 *inDataLen*, U8 \**outData*, U16 \**outDataLen*)  
Invokes data reflection APDU (facilitates link testing). No check of data payload returned

### Parameters

- [in] `inData`: The data to be sent to the Secure Element
- [in] `inDataLen`: The length of `inData`
- [inout] `outData`: IN: caller passes a buffer to hold the data to be returned; OUT: contains the returned data
- [inout] `outDataLen`: IN: length of the `outData` buffer passed; OUT: the number of bytes returned in `outData`

#### Return Value

- `HLSE_SW_OK`: Upon successful execution
- `HLSE_ERR_API_ERROR`: Invalid function arguments

`HLSE_RET_CODE HLSE_DbgReset` (void)  
Resets the Secure Module to the initial state.

#### Return Value

- HLSE\_SW\_OK: Upon successful execution

HLSE\_RET\_CODE **HLSE\_NormalizeECCSignature** (U8 \*signature, U16 signatureLen, U8 \*normalizedSignature, U16 \*normalizedSignatureLen)

The purpose of this function is to turn the proprietary ECDSA signature format - that may be returned by the applet - into a normalized ASN.1 format.

#### Parameters

- [in] signature: buffer containing the ECDSA signature in the applet specific format; OUT: Signature compliant to ASN.1
- [in] signatureLen: length of ECDSA signature length
- [inout] normalizedSignature: IN: caller passes a buffer to hold the data to be returned; OUT: contains the returned data
- [inout] normalizedSignatureLen: IN: length of the outData buffer passed; OUT: the number of bytes returned in outData

#### Return Value

- HLSE\_SW\_OK: upon successful execution
- HLSE\_ERR\_API\_ERROR: Invalid function arguments

## 8.5 A71CH Legacy Configure Tool

### 8.5.1 Introduction

The A71CH Configure Tool is a command line tool that supports the insertion of credentials into the A71CH. It can also report on the value and status of the stored credentials and on the status of the device. The tool is provided in source code (`.../hostlib/a71ch/app`) and can be deployed in one of the following configurations:

- Installed on a development PC communicating over TCP/IP with the embedded target
- Standalone on an embedded target

In *Tool deployment* we go into more detail on this.

Simply invoking the tool in standalone mode on an MCIMX6UL-EVKB board results in the following output (some output edited away):

```
root@imx6ulevk:~# ./a71chConfig_i2c_imx
a71chConfig (Rev 1.00) .. connect to A71CH. Chunksize at link layer = 256.
...
Applet-Rev:SecureBox-Rev : 0x0131:0x0000

Usage: a71chConfig
→ [apdu|debug|erase|gen|info|interactive|lock|rcrt|scp|set|wcr|help] <OptArg>
 apdu -cmd <hexval> -sw <hexval>
 debug [permanently_disable_debug|reset]
 ecrt -x <int>
 erase [cnt|pair|pub|sym] -x <int>
 gen pair -x <int>
 get pub -c <hex_value> -x <int> -k <keyfile.pem>
 info [all|cnt|device|objects|pair|pub|status]
 info gp -h <hexvalue_offset> -n <segments>
 interactive
```

(continues on next page)

(continued from previous page)

```
lock [pair|pub] -x <int>
lock gp -h <hexvalue_offset> -n <segments>
lock inject_plain
obj erase -x <int>
obj get -x <int> [-h <hexvalue_offset>] [-s <hexvalue_size>] [-f <data.txt> -t_
↪[hex_16|hex_32]]
obj update -x <int> -h <hexvalue_offset> [-f <data.txt> -t [hex_16|hex_32] | -h
↪<hexvalue_data>]
obj write -x <int> [-f <data.txt> -t [hex_16|hex_32] | -h <hexvalue_data> | -n
↪<segments>]
rcrt -x <int> [-c <certfile.crt>]
refpem -c <hex_value> -x <int> [-k <keyfile.pem>] -r <ref_keyfile.pem>
script -f <script.txt>
scp [put|auth] -h <hexvalue_keyversion> -k <keyfile>
set gp -h <hexvalue_offset> -h <hexvalue_data>
set pair -x <int> [-k <keyfile.pem> | -h <hexvalue_pub> -h <hexvalue_priv>] [-w
↪<hexvalue_wrap_key>]
set pub -x <int> [-k <keyfile.pem> | -h <hexvalue>] [-w <hexvalue_wrap_key>]
set [cfg|cnt|sym] -x <int> -h <hexvalue> [-w <hexvalue_wrap_key>]
transport [lock|unlock -h <hexvalue_tpkey>]
ucrt -x <int> [-c <certfile.crt> | -h <hexvalue_data> | -p <certfile.pem>]
wcrt -x <int> [-c <certfile.crt> | -h <hexvalue_data> | -p <certfile.pem>] [-n
↪<padding-segments>]

```

The tool provides an overview of the available command line options. We'll go into more detail on the syntax in [Command reference](#).

The easiest way to get familiar with the A71CH configure tool is to open it in interactive mode. Be sure to connect to an A71CH with the Debug Mode still available so you can easily revert to the initial state of the component. The following captures a session with a brand new A71CH with the Debug Mode active:

```
root@imx6ulevk:~/axHostSw/linux# ./a71chConfig_i2c_imx interactive
a71chConfig (Rev 1.00) .. connect to A71CH. Chunksize at link layer = 256.
I2CInit: opening /dev/i2c-1
I2C driver: PEC flag cleared
I2C driver supports plain i2c-level commands.
I2C driver supports Read Block.
SCI2C_ATR=0xB8.03.11.01.05.B9.02.01.01.BA.01.01.BB.0C.41.37.30.30.35.43.47.32.34.32.
↪52.31.BC.00.
HostLib Version : 0x0130
Applet-Rev:SecureBox-Rev : 0x0131:0x0000
>>> info device
A71CH in Debug Mode Version (SCP03 is not set up)
selectResponse: 0x0131
transportLockState: 0x03 (Transport Lock NOT YET set)
injectLockState: 0x02 (Unlocked)
gpStorageSize: 4096
uid (LEN=18):
47:90:51:68:47:91:12:10:23:41:00:53:66:96:47:51:48:12
>>> info pair
Public Keys from ECC key pairs:
 idx=0x00 n.a.
 idx=0x01 n.a.
 idx=0x02 n.a.
 idx=0x03 n.a.
>>> gen pair -x 0
```

(continues on next page)

(continued from previous page)

```
>>> info pair
Public Keys from ECC key pairs:
 idx=0x00 ECC_PUB (LEN=65):
04:0A:81:86:1D:0C:E6:F6:E4:57:65:8B:51:92:E9:D1:CB:AF:96:12:C6:71:FB:79:F1:3D:C9:64:4D:56:CC:87:
2E:8C:32:9B:0A:F8:BB:4B:79:56:7D:F0:9D:C2:D2:B8:96:E0:04:B7:D9:50:F5:EC:C2:50:99:25:6B:5B:4B:E1:
3B
 idx=0x01 n.a.
 idx=0x02 n.a.
 idx=0x03 n.a.
>>> quit
root@imx6ulevk:~/axHostSw/linux#
```

## 8.5.2 Usage modes

The A71CH Configure Tool can be used in:

- Interactive mode. The tool opens a communication session with the A71CH, the user can issue configure commands in this session. The syntax to be used is identical to the syntax used in the command line mode.
- Command line mode: passing parameters as command line arguments. Each invocation of the tool establishes a new communication session between Host and A71CH.
- Batch file mode: this is a special variant of the command line mode where multiple configure commands are bundled in a file that is passed as a command line argument. All commands contained in the file are handled in the same communication session between Host and A71CH.

---

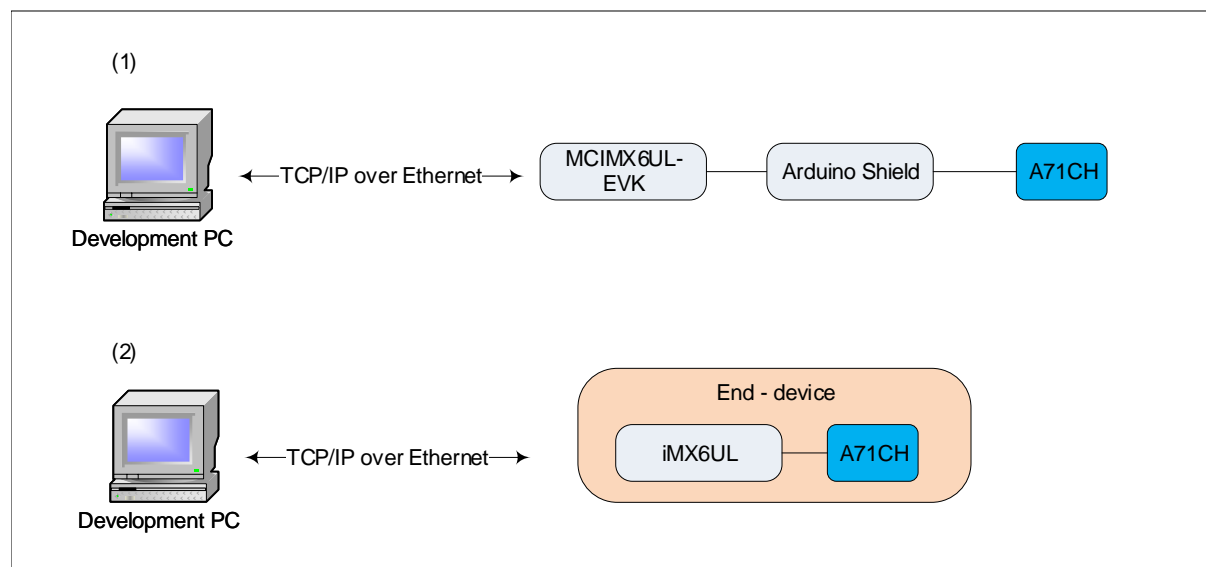
**Note:** On POSIX platforms like LINUX or Cygwin the interactive mode supports simple command line completion and command history (navigateable with the up and down arrows). It also stores a list of executed commands in a file called 'a71chConfigCmdHistory.txt'.

---

## 8.5.3 Tool deployment

### HW Setup for iMX

The HW setup, when using the Configure tool is illustrated the following figure. In case (1) the A71CH has not been integrated into an end-device yet. In case (2) the A71CH is already integrated into the end-device (e.g. an IoT Appliance)

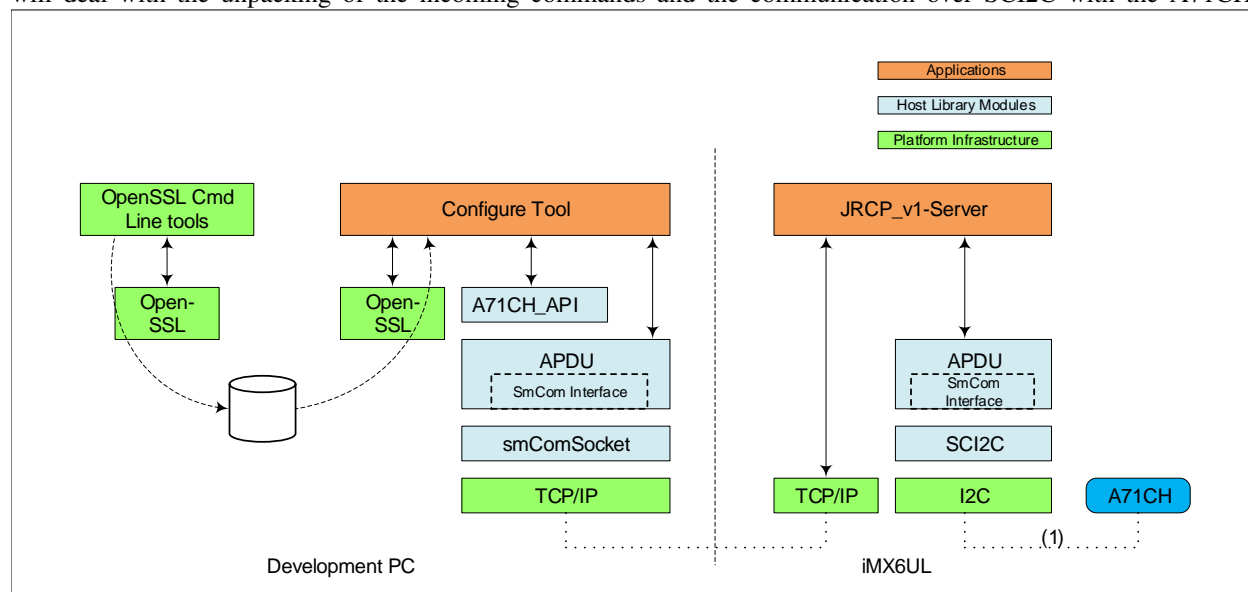


## HW Setup for Kinetis

For running the configure tool with a Kinetis system, USB-VCOM Interface to PC is used. In this combination the VCOM Application needs to be running on kinetis. For more information, see [SW layers and communication for Kinetis](#).

## SW layers and communication for iMX

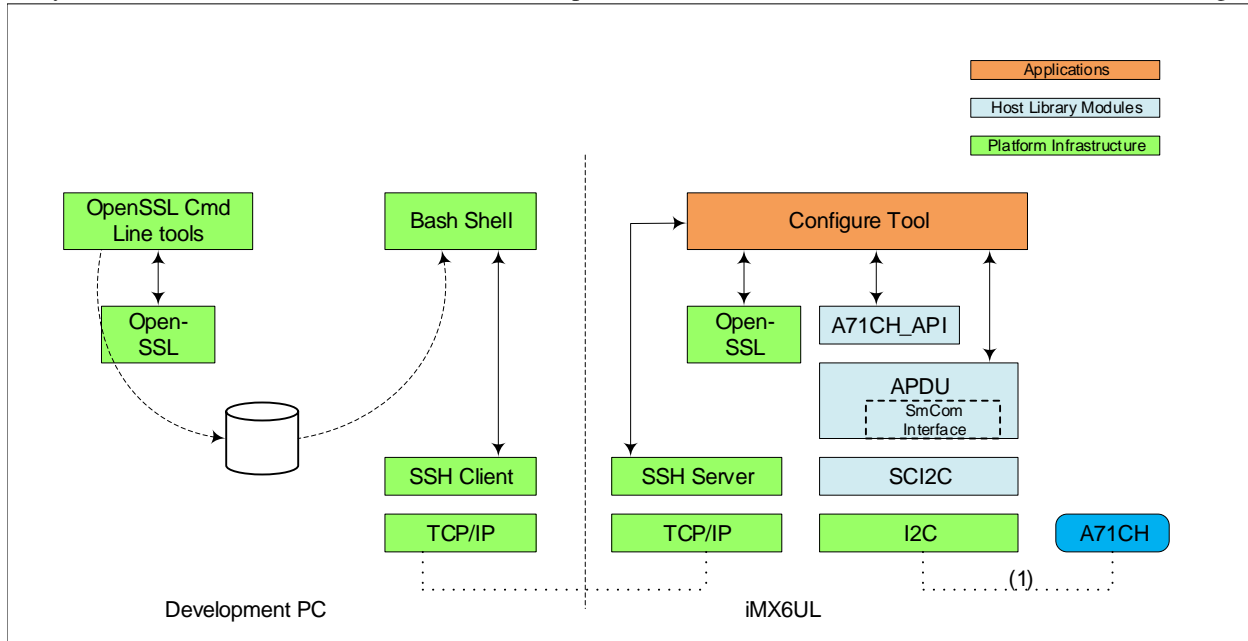
In case the Configure Tool is installed on a development PC, the iMX6UL must run an RJCT-server process that will deal with the unpacking of the incoming commands and the communication over SCI2C with the A71CH.



**Note:** Refer to [JRCP\\_v1 Server](#) for more information on the RJCT server.



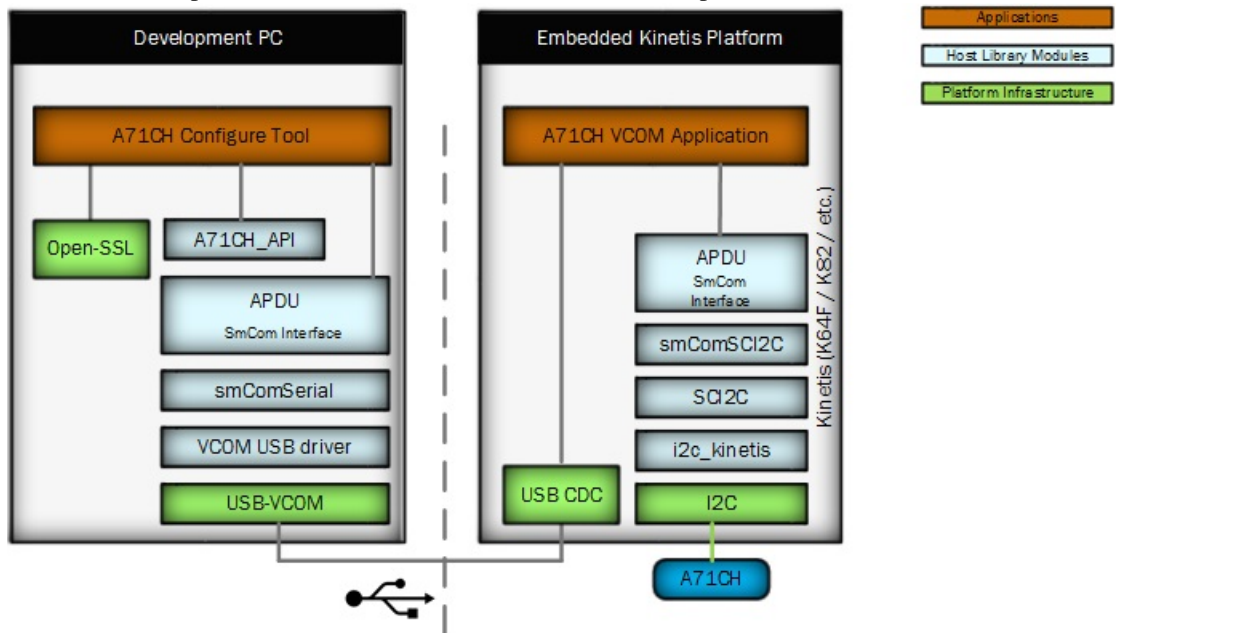
In case the Configure Tool is installed on the embedded target, a development PC will typically be used to run a console that provides access via SSH to the embedded target.



## SW layers and communication for Kinetis

For Kinetis based embedded systems, the configuration tool can only be run from the PC. Also, the configuration tool is only compiled with OpenSSL (not with mbedTLS). VCOM needs to be running on the Kinetis platform and the communication between HostPC and Kinetis happens over USB VCOM.

The Kinetis platform will take care of SCI2C protocol communication with the A71CH.



## 8.5.4 Command reference

### Overall introduction

A command has the following general structure: a mandatory command name `<cmd-n>` is followed by an optional command qualifier `<cmd-q>`, followed by ‘0 to n’ (option, value) pairs.

```
<cmd-n> [<cmd-q>] [-option <option-value>]*
```

The command names `<cmd-n>` are further listed and explained in detail in the remainder of this section.

```
<cmd-n> = {apdu, debug, erase, gen, info, ...}
```

Legal values for command qualifiers `<cmd-q>` depend on the actual command name `<cmd-n>`.

```
<cmd-q> = {cnt, gp, pair, pub, sym, ...}
 cfg = configure key
 cnt = monotonic counter
 gp = general purpose data
 pair = ECC key pair
 pub = ECC public key
 sym = Symmetric secret
```

```
<cmd-q> = {permanently_disable_debug, reset, all, ...}
```

Legal (option, value) pairs again depend on the preceding `<cmd-n>` or `<cmd-n> <cmd-q>`. The order of the (option, value) pairs after the `<cmd-n>` or `<cmd-n> <cmd-q>` needs to be strictly respected. The type of the value, can be any of the following

```
<hexvalue> = [0-9A-F][0-9A-F]([0-9A-F][0-9A-F])*
 examples of legal hexvalue's are
 0A0B0C0D
 00112233445566778899AABBCCDDEEFF
 the following hexvalue's are not allowed
 0x0A0B0C0D # leading '0x' decorator is not supported
 0A1 # odd number of ascii characters is not supported

<int> = integer (currently only positive integers are supported)

<filename> = further explained with the individual commands
```

### apdu

```
apdu -cmd <hexvalue> -sw <hexvalue>
```

The apdu command allows to exchange an APDU (in ‘raw’ format) between the Host and the A71CH. It’s mandatory to specify the expected status word that will be returned by the A71CH, if the actual returned status word is different this will be flagged as an execution error.

---

**Note:** This low level command can be used to extend the functionality of the Config Tool. In order to use this command one needs to consult the A71CH APDU specification. This command is not required for normal provisioning use cases.

---

In the following example the host requests the A71CH the SHA256 value of “F0F1F2F3”. The APDU command and response are printed on the console. The last two byte contained in the response

```
>>> apdu -cmd 8096000004F0F1F2F300 -sw 9000
cmd (LEN=10):
8096000004F0F1F2F300
rsp (LEN=34):
FEA4CE6719F1FDB6D2E30CFB86C2E797DBD4A3247FF2B0EFC15A814C5B25C75E9000
```

## connect

```
connect [close|open]
```

The `connect` command allows to close or re-open the connection with an attached secure element. This command can be used in an interactive workflow where several instance of an A71CH are being configured. Before detaching a configured A71CH one calls `connect close`; after attaching another A71CH one calls `connect open`.

In the following example a connection is opened.

```
>>> connect open
I2CInit: opening /dev/i2c-1
I2C driver: PEC flag cleared
I2C driver supports plain i2c-level commands.
I2C driver supports Read Block.
```

## debug

```
debug [permanently_disable_debug|reset]
```

The `debug` command can be used to permanently switch of the Debug Mode of the A71CH (the Debug Mode of the A71CH is a convenience mode that can be used during product development). It can also be used - assuming the Debug Mode is still on - to bring the A71CH back to its initial state.

---

**Note:** Issuing a debug reset also erases all stored credentials.

---

In the following example a debug reset is issued.

```
>>> debug reset
```

## ecrt

The `ecrt` command erases a certificate from the GP storage area by index.

```
ecrt -x <int>
```

---

**Note:** The valid index range for certificates is limited only by memory size.

---

In the following `ecrt` example the certificate at index 3 is erased from the A71CH.

```
>>> ecrt -x 3
```

### erase

The `erase` command erases (deletes the value) of the specified stored credential. A locked credential can not be erased. Erasing a monotonic counter value is only possible when the Debug Mode of the A71CH is available.

```
erase [cnt|pair|pub|sym] -x <int>
```

In the following example the ECC key pair stored on index 0 is erased.

```
erase pair -x 0
```

### gen

The `gen` command makes the A71CH create a valid ECC key pair on the indicated index.

```
gen pair -x <int>
```

In the following example a new ECC keypair is created and stored on index 1

```
gen pair -x 1
```

### get

The `get` command retrieves the public key value from either a public key or key pair at the index passed as argument and stores it - in pem format - in a file provided as argument.

---

**Note:** The parameter passed after the `c` option represents the key type and can be either 0x10 for public pair or 0x20 for public key.

---

```
get pub -c <hex_value> -x <int> -k <keyfile.pem>
```

In the following example the ECC public key stored at index 0 is stored to PEM file `keyfile.pem`

```
get pub -c 20 -x 0 -k keyfile.pem
```

### info

The `info` command can be used to echo the value and/or status of the A71CH or its stored credentials to the console. Issuing an 'info all' will echo the same information as issuing 'info device', 'info cnt', 'info pair', 'info pub', 'info gp -h 0000 -n <all>' in sequence. The value of secret credentials like the private part of a keypair, a symmetric key or a configuration key can not be retrieved from the A71CH. The 'info status' command will report on the Initialized/Empty and Locked/Open status of all credentials. It's possible to echo the value of consecutive 32 byte data segments from general purpose data storage by specifying the hexadecimal offset (specified with 4 hexadecimal digits) into the data store and the amount of segments to display.

```
info [all|device|cnt|pair|pub|sym|status]
info gp -h <hexvalue_offset> -n <segments>
```

In the following example the credential status is requested. The output corresponds to the status of a new device.

```
>>> info status
SCP03 is Not enabled
Key Pair status:
 Index=0: Empty Open
 Index=1: Empty Open
 Index=2: Empty Open
 Index=3: Empty Open
Public Key status:
 Index=0: Empty Open
 Index=1: Empty Open
 Index=2: Empty Open
Config Key status:
 Index=0: Empty Open
 Index=1: Empty Open
 Index=2: Empty Open
Sym Secret status:
 Index=0: Empty Open
 Index=1: Empty Open
 Index=2: Empty Open
 Index=3: Empty Open
Counter status:
 Index=0: Initialized Open
 Index=1: Initialized Open
Certificate Objects:
 0 Absolute offset = 0x00 Actual Size = 0x313
 1 Absolute offset = 0x320 Actual Size = 0x313
Data Objects:
 0 Absolute offset = 0x640 Actual Size = 0x09
 1 Absolute offset = 0x660 Actual Size = 0x09
General Purpose Storage status:
 Offset=0x0000: Open Offset=0x0020: Open Offset=0x0040: Open ↵
↵Offset=0x0060: Open
 Offset=0x0080: Open Offset=0x00A0: Open Offset=0x00C0: Open ↵
↵Offset=0x00E0: Open
 Offset=0x0100: Open Offset=0x0120: Open Offset=0x0140: Open ↵
↵Offset=0x0160: Open
 Offset=0x0180: Open Offset=0x01A0: Open Offset=0x01C0: Open ↵
↵Offset=0x01E0: Open
 Offset=0x0200: Open Offset=0x0220: Open Offset=0x0240: Open ↵
↵Offset=0x0260: Open
 Offset=0x0280: Open Offset=0x02A0: Open Offset=0x02C0: Open ↵
↵Offset=0x02E0: Open
 Offset=0x0300: Open Offset=0x0320: Open Offset=0x0340: Open ↵
↵Offset=0x0360: Open
 Offset=0x0380: Open Offset=0x03A0: Open Offset=0x03C0: Open ↵
↵Offset=0x03E0: Open
```

In the following example the contents from two 32 byte data segments is requested starting from general purpose storage offset 0x0010:

```
>>> info gp -h 0010 -n 2
GP Storage Data (2 segments from offset 0x0010):
 0x0010 (LEN=32): ↵
↵00
 0x0030 (LEN=32): ↵
↵00
```

## interactive

Used to start the interactive mode from the command line

## lock

The `lock` commands allows to lock individual credentials (ECC public keys and ECC key pairs). It allows to lock data segments of 32 byte in general purpose storage (on offsets that are multiples of 0x0020). It's also possible to forbid the injection of unwrapped ECC public keys, ECC key pairs and symmetric secrets at the device level.

```
lock [pair|pub] -x <int>
lock gp -h <hexvalue_offset> -n <segments>
lock inject_plain
```

The following example locks the ECC key pair at index 0

```
>>> lock pair -x 0
```

The following example locks 2 data segments of 32 byte in general purpose data storage starting from offset 0x0060

```
>>> lock gp -h 0060 -n 2
```

## obj erase

The `obj erase` command erases the object at the provided index.

```
obj erase -x <int>
```

---

**Note:** Upon erasing an object it cannot be reconstructed.

---

In the following `obj erase` example the object at index 0 is erased.

```
>>> obj erase -x 0
```

## obj get

The `obj get` command gets the value of a data object, it retrieves the data from a specific offset within the data object (fetching the specified amount of byte). Optionally, the data is written to file. The type file could be 16 or 32 bytes at a line. If no type is specified the default would be 32 bytes.

```
obj get -x <int> [-h <hexvalue_offset>] [-s <hexvalue_size>] [-f <data.txt> -t [hex_
↪16|hex_32]]
```

---

**Note:** The offset is relative to the start location of the object and must be specified as a 4 digit hexadecimal value.

---

In the following `obj get` example the value of the object at index 0 is read out.

```
>>> obj get -x 0 -h 0000 -s 0009
>>> 112233445566778899
```

## obj update

The `obj update` command updates the value of a data object. It updates the data relative to an internal offset passed as a parameter. The data can be passed on the command line or be contained in a file.

```
obj update -x <int> -h <hexvalue_offset> [-f <data.txt> -t [hex_16|hex_32] | -h
↪<hexvalue_data>]
```

**Note:** The data in the file must be binary and not textual. An object must already exist at the specified index. If data is read from file it can be set with lines in length of 16 or 32 bytes (i.e. `hex_16` or `hex_32`). The default value is lines of 32 bytes.

In the following `obj update` example the value of the object at index 0 is updated.

```
>>> obj update -x 0 -h 0000 -h 998877665544332211
```

## obj write

The `obj write` command creates an object. The value of the object to be created can be passed on the command line or contained in a file. When using the `-n` option the requested segments will be reserved for the data object and filled with zeros. If data is read from file it can be set with lines in length of 16 or 32 bytes (i.e. `hex_16` or `hex_32`). The default value is lines of 32 bytes.

```
obj write -x <int> [-f <data.txt> -t [hex_16|hex_32] | -h <hexvalue_data> | -n
↪<segments>]
```

In the following `obj write` example an zero filled object is created at index 0 with a size of 5 segments.

```
>>> obj write -x 0 -n 5
```

## rcrt

The `rcrt` command reads a certificate from the GP storage area by index. Optionally, the command can save the certificate read to a CRT file.

```
rcrt -x <int> [-c <certfile.crt>]
```

**Note:** The certificate data will be presented whether it was written to a file or not. The valid index range for certificates is limited only by memory size.

In the following `rcrt` example the certificate at index 3 is read from the A71CH, upon success it is also written to a CRT file.

```
>>> rcrt -x 3 -c certificate.crt
CER_DATA (LEN=520):
30820204308201A9020900CFD5820FFEC40937300A06082A8648CE3D04030230
8189310B30090603550406130242453116301406035504080C0D566C61616D73
42726162616E74310F300D06035504070C064C657576656E3111300F06035504
0A0C084E58502D44656D6F31163014060355040B0C0D4E58502D44656D6F2D55
6E6974310D300B06035504030C0464656D6F3117301506092A864886F70D0109
```

(continues on next page)

(continued from previous page)

```
01160864656D6F406E7870301E170D3135313230373130353132395A170D3136
313230363130353132395A308188310B30090603550406130242453116301406
035504080C0D566C61616D7342726162616E74310F300D06035504070C064C65
7576656E310E300C060355040A0C05697063616D31123010060355040B0C0969
7063616D556E69743112301006035504030C09697063616D44656D6F31183016
06092A864886F70D0109011609697063616D406E78703059301306072A8648CE
3D020106082A8648CE3D03010703420004DB4CDB6C5A96C1615895095222AA0E
A3BC6F9E714D6438F0B120D691F18D7E7410EE04BE71D33A2D8B2D3B66F7174A
9654536965AFD2ABADB55269C6A6C0085E300A06082A8648CE3D040302034900
304602210083AA91AE33396825D560390952AEE91C64814C7CA681BA50589558
D681F974270221009BA1CF31A823B96C391E3C4F839666AECE9949639D796B24
A5B987A92E6F1CFA
```

## refpem

**Note:** The reference keys created by the `refpem` command are **only** compatible with the A71CH OpenSSL Engine based upon the A71CH Legacy API. The A71CH OpenSSL Engine based upon the SSS API use a different reference key format, these keys must be created with the `sscli` tool.

The `refpem` command allows to create A71CH OpenSSL Engine specific reference pem files. It can be used in a mode that fetches the public key value from the attached A71CH:

```
refpem -c <hex_value> -x <int> -r <ref_keyfile.pem>
```

Or it can be used in a ‘not-connected’ mode that fetches the public key value from a pem file (containing an EC key pair) supplied as an argument.

```
refpem -c <hex_value> -x <int> -k <keyfile.pem> -r <ref_keyfile.pem>
```

The value following the `-c` switch must be either 10 (create a reference to a key pair) or 20 (create a reference to a public key). The value following the `-x` switch is the storage index of either key pair or public key.

The following command creates a reference pem file ‘my\_ref\_keyfile.pem’ referring to a keypair stored at index 1.

```
refpem -c 10 -x 1 -r my_ref_keyfile.pem
```

## script

The `script` command can be used to issue the Configure tool commands contained in a file.

```
script -f <script.txt>
```

An example of script file (`script_example.txt`)

```
root@imx6ulevk:~# cat script_example.txt
Simple example script
info pair
gen pair -x 0
info pair # This will illustrate a key pair was created
```

The following example issues the commands contained in the script file above (`script_example.txt`)



```
>>> script -f script_example.txt
>> # Simple example script

>> info pair

Public Keys from ECC key pairs:
 idx=0x00 n.a.
 idx=0x01 n.a.
>> gen pair -x 0

>> info pair # This will illustrate a key pair was created

Public Keys from ECC key pairs:
 idx=0x00 ECC_PUB (LEN=65):
04:A4:B3:3B:A3:D4:23:BD:19:C3:CB:20:DB:6F:D3:80:46:73:06:56:2F:83:B2:B1:AE:86:9A:EF:E9:7A:62:A3:
04:E7:C1:42:31:97:D5:19:5A:80:27:74:DC:20:EC:B7:93:9B:E5:C1:22:22:6B:E3:49:A4:FB:3A:5C:26:08:85:
B5
 idx=0x01 n.a.
```

## scp

The `scp` command can be used to write a set of SCP03 keys to the A71CH ('`scp put ...`') or to establish an active SCP03 channel between Host and A71CH ('`scp auth ...`'). The '`scp clear_host`' command will force the Host to issue commands in the clear again.

```
scp [put|auth] -h <hexvalue_keyversion> -k <keyfile>
scp clear_host
```

An example of a keyfile containing a set of SCP03 keys:

```
root@imx6ulevk:~# cat scp_keyfile_example.txt
This is a comment, empty lines and comment lines allowed.
ENC AA112233445566778899AABBCCDDEEFF # Trailing comment
MAC BB112233445566778899AABBCCDDEEFF # Optional trailing comment
DEK CC112233445566778899AABBCCDDEEFF # Optional trailing comment
```

## set

The `set` command can be used to set a credential stored on the A71CH to a specific value.

---

**Note:** The `set gp` command can only be used to set a maximum of 32 byte of data at a time.

The value of a key pair or public key can either be passed as command line parameters or be contained in a pem-file (containing an EC key pair).

The command line value of the private key (set by the `set pair` command) can be either in the clear or wrapped with the Configuration key stored at index 1. Wrapping is according to [RFC3394](#).

The command line value of the public key (set by the `set pub` command) can be either in the clear or wrapped with the Configuration key stored at index 2. In case [RFC3394](#) wrapping is applied the first byte of the public key (the one indicating the public key format) is removed before applying wrapping.

The value of the configure key, the monotonic counter or the symmetric secret can only be passed explicitly as a command line parameter. The configure and symmetric keys can also be set wrapped (with the stored value of the key) according to [RFC3394](#).

Whether an argument is wrapped is implicit in the lenght of the provided argument.

```
set gp -h <hexvalue_offset> -h <hexvalue_data>
set pair -x <int> [-k <keyfile.pem> | -h <hexvalue_pub> -h <hexvalue_priv>]
set pub -x <int> [-k <keyfile.pem> | -h <hexvalue>]
set [cfg|cnt|svm] -x <int> -h <hexvalue>
```

The following example writes 5 byte of data at offset 0004 into the General Purpose data store. The data written (4137314348) is the equivalent of the ASCII encoding of the string 'A71CH'. The command itself is preceded and followed by an info statement covering the general purpose storage segment of interest.

```
>>> info gp -h 0000 -n 1
GP Storage Data (1 segments from offset 0x0000):
 0x0000 (LEN=32):

 00

>>> set gp -h 0004 -h 4137314348

>>> info gp -h 0000 -n 1
GP Storage Data (1 segments from offset 0x0000):
 0x0000 (LEN=32):

 00000000413731434800
```

The following example set the key pair at index 1 from the value contained in file `keyfile_ecc_nist_256_1.pem`. The command itself is preceded and followed by an info statement on the stored key pairs.

```
>>> info pair
Public Keys from ECC key pairs:
 idx=0x00 n.a.
 idx=0x01 n.a.
>>> set pair -x 1 -k keyfile_ecc_nist_256_1.pem
ECCPrivateKey (LEN=32):
21:AF:C1:1E:F5:64:61:3D:2E:96:4D:8B:93:19:CC:AB:38:E0:7A:6E:35:3A:21:A3:D1:69:8B:19:13:DF:1D:FF

ECCPublicKey (LEN=65):
04:74:E2:1E:54:6C:C1:9E:31:58:55:B6:D5:45:D3:0D:3F:48:79:D4:64:5D:3F:67:73:75:FB:0B:2C:80:43:1E:
8D:34:95:71:0E:71:E1:E3:F8:93:62:75:B4:AC:F1:52:E3:DE:55:CC:1D:86:5E:B0:D1:22:A8:CF:35:EC:47:31:
F8
>>> info pair
Public Keys from ECC key pairs:
 idx=0x00 n.a.
 idx=0x01 ECC_PUB (LEN=65):
04:74:E2:1E:54:6C:C1:9E:31:58:55:B6:D5:45:D3:0D:3F:48:79:D4:64:5D:3F:67:73:75:FB:0B:2C:80:43:1E:
8D:34:95:71:0E:71:E1:E3:F8:93:62:75:B4:AC:F1:52:E3:DE:55:CC:1D:86:5E:B0:D1:22:A8:CF:35:EC:47:31:
F8

The value contained in file keyfile_ecc_nist_256_1.pem is
```

```
$ cat keyfile_ecc_nist_256_1.pem
-----BEGIN EC PRIVATE KEY-----
MHcCAQEEICGvwr71ZGE9lpZNi5MzZKs44HpuNToho9FpixkT3x3/oAoGCCqGSM49
AwEHoUQDQgAEoIEvGzBnJfFYVbbVVRdMNP0h51GRdP2dzdfsLLIBDH001XE0ceHj
+JNidbSs8VLj3lXMHYZesNEiqM817EcX+A==
-----END EC PRIVATE KEY-----
```

The following example sets the public key at index 0 to the provided public key value (in the clear, ANSI X9.62 uncompressed format). The command itself is preceded and followed by an info statement on the stored public key.

```
>>> info pub
Public Keys:
 idx=0x00 n.a.
 idx=0x01 n.a.
>>> set pub -x 0 -h_
↪043802B1164C30860AC913F5F997B84158C40CFFCC1D3A4359BC22574A4FC95E628933A9E95820AD6B96A1DA106BDD5D6A
>>> info pub
Public Keys:
 idx=0x00 ECC_PUB (LEN=65):
04:38:02:B1:16:4C:30:86:0A:C9:13:F5:F9:97:B8:41:58:C4:0C:FF:CC:1D:3A:43:59:BC:22:57:4A:4F:C9:5E:
62:89:33:A9:E9:58:20:AD:6B:96:A1:DA:10:6B:DD:5D:6A:8E:55:6A:78:AE:95:9C:59:33:6F:E5:3E:3A:1D:9E:
D4
 idx=0x01 n.a.
```

The following example sets the monotonic counter at index 0 to 00E0. The command itself is preceded and followed by an info statement on the stored monotonic counters.

```
>>> info cnt
Monotonic counter values:
 idx=0x00 0x00000000
 idx=0x01 0x00000000
>>> set cnt -x 0 -h 000000E0
>>> info cnt
Monotonic counter values:
 idx=0x00 0x000000E0
 idx=0x01 0x00000000
```

## transport

The `transport lock` command can be used to enable the transport lock on the A71CH. To disable the transport lock one needs to pass the transport key as an option value to the `transport unlock` command.

---

**Note:** A precondition to enable the transport lock is that the Transport Configuration key has been set: use ‘set cfg -x 0 -h <hexvalue\_tpkey>’ to achieve this. Furthermore the transport lock / unlock cycle can only be initiated once.

---

```
transport [lock|unlock -h <hexvalue_tpkey>]
```

The following example sets the Transport Configuration key, locks the device and finally unlocks the device. The `info device` command is used to illustrate the value of the `transportLockState` of the device.

```
>>> info device
...
transportLockState: 0x03 (Transport Lock NOT YET set)
...
>>> set cfg -x 0 -h AA112233445566778899AABBCCDDEEFF
>>> transport lock
>>> info device
...
transportLockState: 0x01 (Transport Lock is set)
...
>>> transport unlock -h AA112233445566778899AABBCCDDEEFF
>>> info device
A71CH in Debug Mode Version (SCP03 is not set up)
selectResponse: 0x0111
```

(continues on next page)

(continued from previous page)

```
transportLockState: 0x02 (Open device, Transport Lock can no longer be set)
injectLockState: 0x02 (Unlocked)
gpStorageSize: 1024
uid (LEN=18):
47:90:70:02:47:91:12:10:20:89:00:50:36:91:64:23:00:00
```

## ucrt

The `ucrt` command updates a certificate to the GP storage area by index. The certificate can be provided as raw data (-h option), as a file in PEM format (-p option) or as a file in DER format (-c option).

```
ucrt -x <int> [-c <certfile.crt> | -h <hexvalue_data> | -p <certfile.pem>]
```

---

**Note:** In case the certificate to be written is in PEM format it will be stored into the A71CH in DER format. The valid index range for certificates is limited only by memory size.

---

In the following `ucrt` example a certificate contained in a PEM file (c:\certificate.pem) is stored into the A71CH at index 3.

```
>>> ucrt -x 3 -p c:\certificate.pem
Filename: c:\certificate.pem
Certificate Size (DER format) = 493 byte
```

## wcrt

The `wcrt` command writes a certificate to the GP storage area by index. The certificate can be provided as raw data (-h option), as a file in PEM format (-p option) or as a file in DER format (-c option).

```
wcrt -x <int> [-c <certfile.crt> | -h <hexvalue_data> | -p <certfile.pem>] [-n
↳<padding-segments>]
```

---

**Note:** Writing to an existing index will fail. Use the `ucrt` command to update the certificate (taking into account certificate size constraints) or use the `ecrt` command to erase and then write the new certificate. The valid index range for certificates is limited only by memory size. Using padding segments parameter creates an extra place holder for future updates with larger certificates at the same index without the need for erasing it first.

In case the certificate to be written is in PEM format it will be stored into the A71CH in DER format.

The `rcrt` command allows to read out a certificate by index.

In the following `wcrt` example a certificate contained in a PEM file (c:\certificate.pem) is stored into the A71CH at index 3.

```
>>> wcrt -x 3 -p c:\certificate.pem
Filename: c:\certificate.pem
Certificate Size (DER format) = 493 byte
```

### 8.5.5 Not connected mode

When starting up the A71CH Configure Tool it is possible to indicate no attached A71CH device is required. This is achieved by preceding the command (on the command line only) by the keyword `nc` (not connected).

Currently the only application of this feature is the creation of Reference Pem files where the public key value is contained in a Pem file (containing an EC key pair) passed as argument.

The following command creates a reference pem file 'my\_ref\_keyfile.pem' referring to a public key (stored or to be stored) at index 0 whose value is contained in 'kp\_keyfile.pem'

```
root@imx6ulevk:~# ./a71chConfig_i2c_imx nc refpem -c 20 -x 0 -k kp_keyfile.pem -r my_
↪ref_keyfile.pem
a71chConfig (Rev 0.94) .. NOT connecting to A71CH.
ECCPublicKey (LEN=65):
04:7C:59:16:D4:F5:46:B3:D3:17:20:78:F8:AD:41:84:9A:79:46:6B:5B:0B:FC:39:3D:4C:E1:A8:53:F5:4F:8D:
C2:98:65:F8:84:E9:9E:28:38:09:FF:29:34:B6:97:27:DB:6C:0A:F3:79:B0:D7:2C:16:25:B5:CB:B8:A2:CB:70:
89
```



## APPENDIX

## 9.1 Glossary

Table 1: Glossary

| Term | Definition           |
|------|----------------------|
| EAR  | Early Access Release |
| SE   | Secure Element       |

## 9.2 APDU Commands over VCOM

Sending and Receiving Applet APDU Commands on VCOM using a Serial Terminal Emulator.

### 9.2.1 COM port parameters

| Parameter   | Value        |
|-------------|--------------|
| Baud Rate   | 115200 bit/s |
| Data bits   | 8 Databits   |
| Parity      | No Parity    |
| Stop Bits   | 1 Stop bit   |
| FlowControl | Enable DTR   |

### 9.2.2 VCOM Format

Each APDU/command to be sent over the VCOM bridge has a 4 bytes header, the APDU to send follows as payload. The response contains the same header, followed by the response payload.

| Byte | Description                                                                                                  |
|------|--------------------------------------------------------------------------------------------------------------|
| 1    | Command <ul style="list-style-type: none"><li>00: SoftReset / Fetch ATR</li><li>01: Other commands</li></ul> |
| 2    | Node Address                                                                                                 |
| 3    | Len MSB                                                                                                      |
| 4    | Len LSB                                                                                                      |

## 9.2.3 Example Commands

The example shows the commands strings to be sent over VCOM and some parsing of the commands/responses

### Soft Reset/ATR Response

- Soft Reset / Request ATR .

| Field             | Bytes in Hex                                                                                                            |
|-------------------|-------------------------------------------------------------------------------------------------------------------------|
| Command           | 00 00 00 00                                                                                                             |
| VCOM Header       | –                                                                                                                       |
| APDU              | –                                                                                                                       |
| Response          | 00 00 00 23 00 A0 00 00 03 96 04 03 E8 00 FE 02 0B 03 E8 08 01<br>00 00 00 00 64 00 00 0A 4A 43 4F 50 34 20 41 54 50 4F |
| VCOM Header       | 00 00 00 23                                                                                                             |
| Atr Re-<br>sponse | -- -- -- -- 00 A0 00 00 03 96 04 03 E8 00 FE 02 0B 03 E8 08 01<br>00 00 00 00 64 00 00 0A 4A 43 4F 50 34 20 41 54 50 4F |

### Select IOT Applet

Select the SE050 IoT applet

| Field               | Bytes in Hex                                                                         |
|---------------------|--------------------------------------------------------------------------------------|
| Command             | 01 00 00 15 00 A4 04 00 0F A0 00 00 03 96 54 53 00 00 00 01 03<br>00 00 00 00        |
| VCOM Header         | 01 00 00 15                                                                          |
| APDU                | -- -- -- -- 00 A4 04 00 0F A0 00 00 03 96 54 53 00 00 00 01 03<br>00 00 00 00        |
| Response            | 01 00 00 09 03 01 00 3F FF 01 0B 90 00                                               |
| VCOM Header         | 01 00 00 09                                                                          |
| APDU Re-<br>sponse  | -- -- -- -- 03 01 00 3F FF 01 0B 90 00                                               |
| applet Ver-<br>sion | appletVersion in TLV with Tag1, (Len=7)<br>-- -- -- -- -- -- -- 03 01 00 3F FF 01 0B |

### Get applet Version

Gets the applet version information.



| Field               | Bytes in Hex                                                                            |
|---------------------|-----------------------------------------------------------------------------------------|
| Command             | 01 00 00 04 80 04 00 20                                                                 |
| VCOM Header         | 01 00 00 04                                                                             |
| APDU                | -- -- -- -- 80 04 00 20                                                                 |
| Response            | 01 00 00 0D 41 82 00 07 03 01 00 3F FF 01 0B 90 00                                      |
| VCOM Header         | 01 00 00 0D                                                                             |
| APDU Re-<br>sponse  | -- -- -- -- 41 82 00 07 03 01 00 3F FF 01 0B 90 00                                      |
| applet Ver-<br>sion | appletVersion in TLV with Tag1, (Len=7)<br>-- -- -- -- -- -- -- -- 03 01 00 3F FF 01 0B |

### GetRandom

Gets x(8) byte random data from the SE050.

| Field              | Bytes in Hex                                          |
|--------------------|-------------------------------------------------------|
| Command            | 01 00 00 0D 80 04 00 49 00 00 04 41 02 00 08 00 00    |
| VCOM Header        | 01 00 00 0D                                           |
| APDU               | -- -- -- -- 80 04 00 49 00 00 04 41 02 00 08 00 00    |
| Response           | 01 00 00 0E 41 82 00 08 B5 47 3C 8C A5 16 AC 31 90 00 |
| VCOM Header        | 01 00 00 0E                                           |
| APDU Re-<br>sponse | -- -- -- -- 41 82 00 08 B5 47 3C 8C A5 16 AC 31 90 00 |
| RandomData         | -- -- -- -- -- -- -- -- B5 47 3C 8C A5 16 AC 31       |

### GetUID

UID is an object with Object ID = 0x7FFF0206.

| Field              | Bytes in Hex                                                                                             |
|--------------------|----------------------------------------------------------------------------------------------------------|
| Command            | 01 00 00 13 80 02 00 00 00 00 00 0A 41 04 7F FF 02 06 43 02 00 12<br>00 00                               |
| VCOM Header        | 01 00 00 13                                                                                              |
| APDU               | -- -- -- -- 80 02 00 00 00 00 00 0A 41 04 7F FF 02 06 43 02 00 12<br>00 00                               |
| Response           | 01 00 00 18 41 82 00 12 04 00 50 01 55 55 55 55 55 55 55 04 FF<br>FF FF FF FF FA 90 00                   |
| VCOM Header        | 01 00 00 18                                                                                              |
| APDU Re-<br>sponse | -- -- -- -- 41 82 00 12 04 00 50 01 55 55 55 55 55 55 55 04 FF<br>FF FF FF FF FA 90 00                   |
| UID                | UID in TLV with Tag1<br>-- -- -- -- -- -- -- -- 04 00 50 01 55 55 55 55 55 55 55 04 FF<br>FF FF FF FF FA |

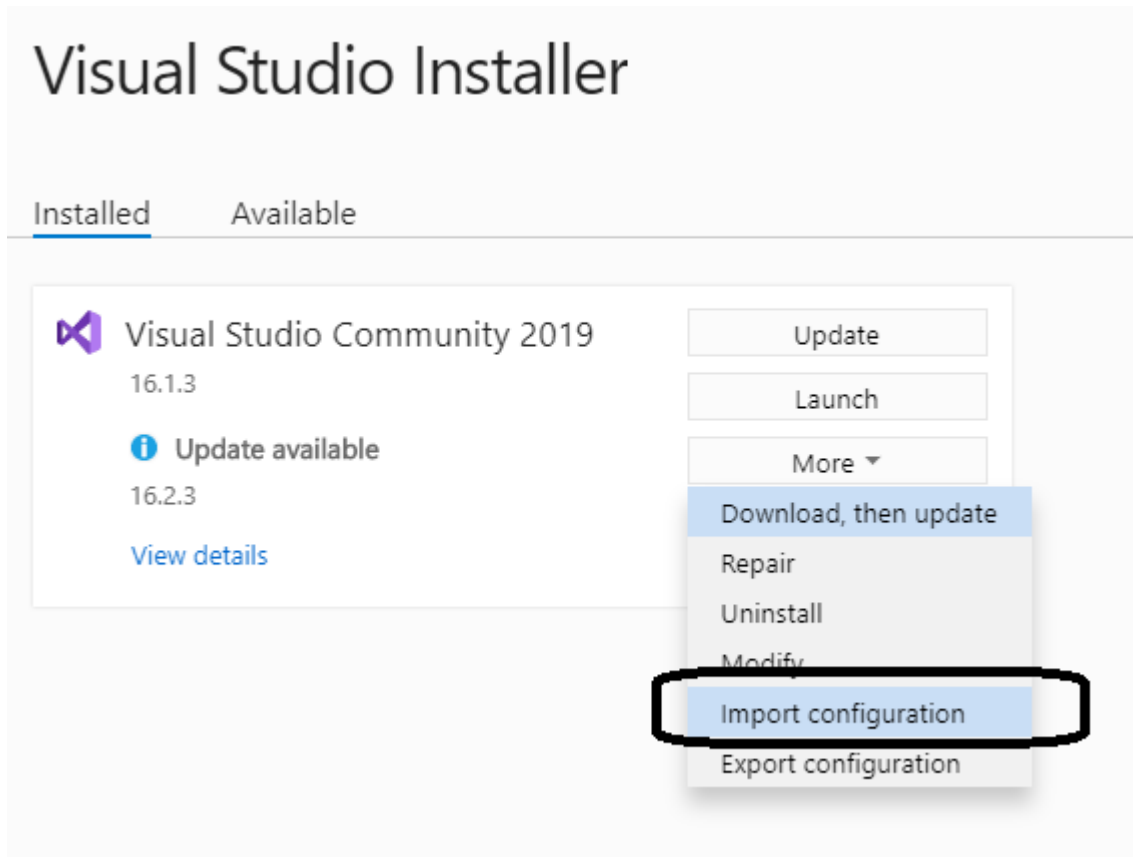
## 9.3 Visual Studio 2019 Setup

### 9.3.1 Prerequisites

Install any edition of Visual Studio 2019, It will install the visual studio installer application.

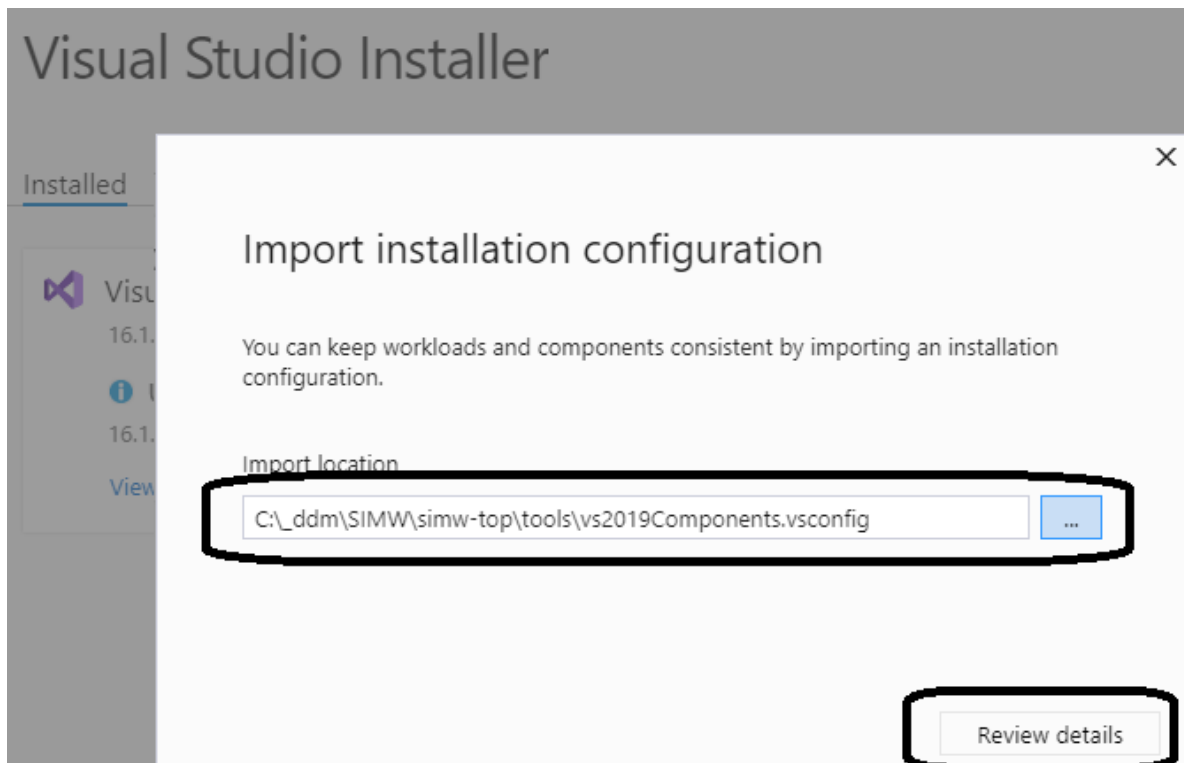
### 9.3.2 Installing the components

- 1) Open Visual Studio Installer, click on more and then import configuration.

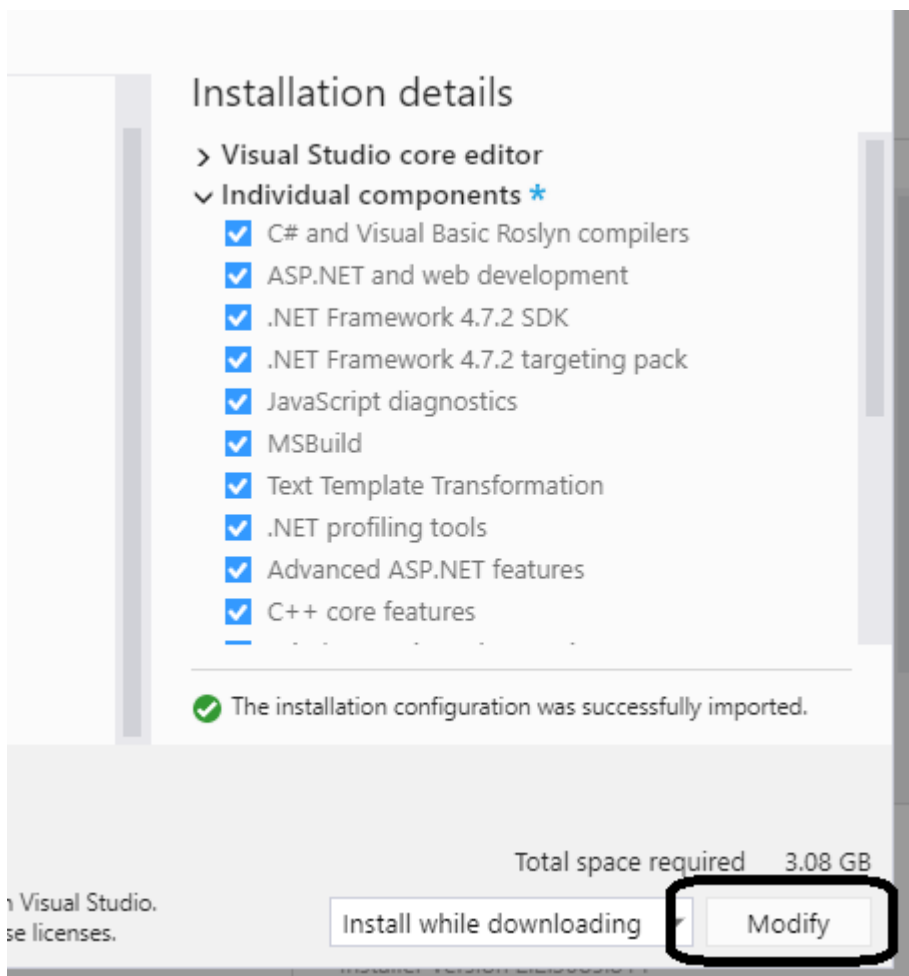


- 2) In the dialogue box give the path to the configuration file present in tools folder.

```
tools/vs2019Components.vsconfig
```



3) click on modify.



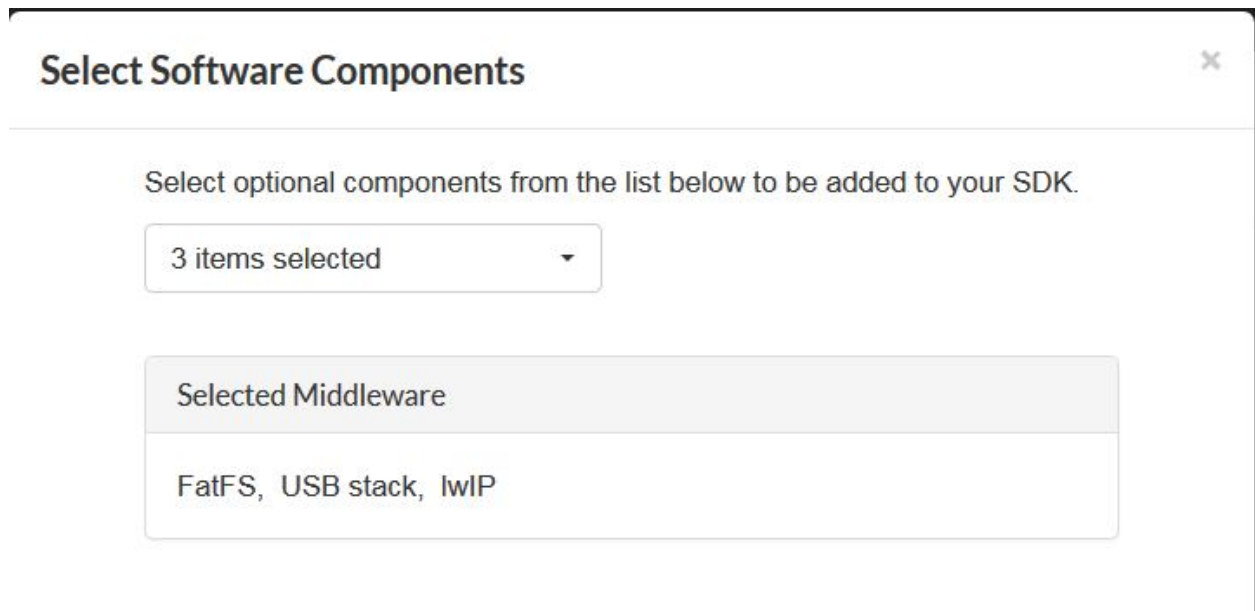
## 9.4 Setting up MCUXpresso IDE

1. Download MCUXpresso from: <https://www.nxp.com/support/developer-resources/software-development-tools/mcuxpresso-software-and-tools/mcuxpresso-integrated-development-environment-ide:MCUXpresso-IDE>
2. For additional help please refer to: [https://www.nxp.com/docs/en/user-guide/MCUXpresso\\_IDE\\_User\\_Guide.pdf](https://www.nxp.com/docs/en/user-guide/MCUXpresso_IDE_User_Guide.pdf)

### 9.4.1 To Download and install MCUXpresso IDE:

To create and install board specific SDK

- Login/create an account on the SDKBuilder website <https://mcuxpresso.nxp.com/en/select> .
- Select your Board i.e. FRDM-K64F or EVKB-IMXRT1050.
- Click on Build MCUXpresso SDK.
- Select Software Components.

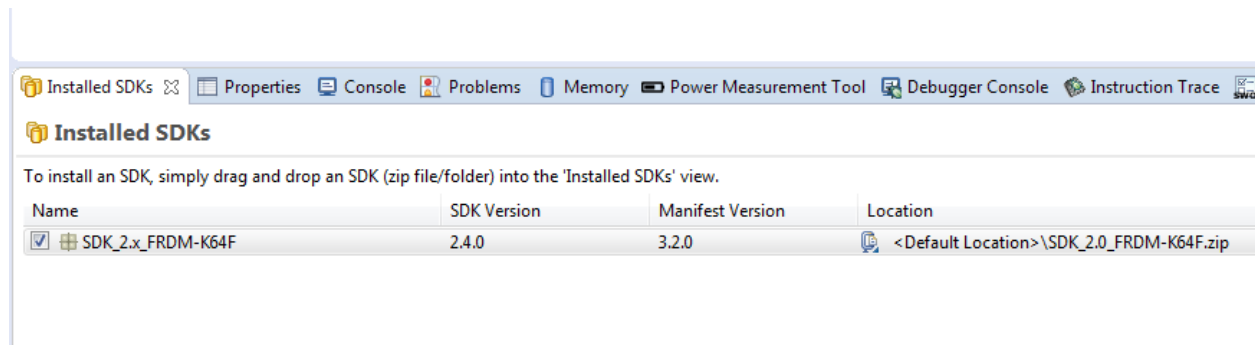


**Note:** MCUXpresso SDK Builder has a limitation with the components getting selected as on today, So to be on the safer side, all the Middleware components have been selected to generate the SDK.

- Download the SDK

## 9.4.2 To Install board specific SDK in MCUXpresso

- In MCUXpresso IDE, install the downloaded SDK by dragging and dropping it into the Installed SDK's tab.



## 9.5 Development Platforms

### 9.5.1 Setup i.MX 8MQuad - MCIMX8M-EVK

This section explains how to create an SD card image (including native compilation tools) and a cross-compilation environment for the MCIMX8M-EVK.

**Note:** Evaluation Board Type

This guidelines refers to the MCIMX8M-EVK evaluation board. Please refer to the bottom of this page for guidelines on connecting the SE050 Arduino shield to the MCIMX8M-EVK

---

### Downloading and Installing Yocto for MCIMX8M-EVK

Please consult first the detailed information that can be found on <https://www.nxp.com>. Download the L4.19.35\_1.1.0\_LINUX\_DOCS documentation package (available on <https://www.nxp.com/design/i-mx-developer-resources/i-mx-software-and-development-tools:IMX-SW> under the header **Overview - Documentation - Linux 4.19.35\_1.1.0 Documentation** and take the **i.MX Yocto Project User's Guide** as a starting point.

Yocto must be installed on a Linux PC (consult the documentation for the Linux distributions officially supported), it's possible to use a Virtual PC environment as e.g. Virtual Box. The Linux PC must have access to the internet to retrieve additional packages, tools and sources.

Having set-up all tools and packages required by Yocto, initialize a local git repository as follows:

```
mkdir -p ~/projects/imx-yocto-bsp-4-19-35-1-1-0
cd ~/projects/imx-yocto-bsp-4-19-35-1-1-0
repo init -u https://source.codeaurora.org/external/imx/imx-manifest -b imx-linux-
↳ warrior -m imx-4.19.35-1.1.0.xml
repo sync
```

Add a custom Yocto layer. This custom Yocto layer is part of the Plug&Trust SW distribution (simw\_top/scripts/yocto/layers/meta-custom.tgz) and must be copied and unpacked into the sources directory created above. Note: this custom layer adds a Python package (func-timeout):

```
cp_
↳ <PlugTrust>/simw_top/scripts/yocto/layers/meta-custom.tgz ~/projects/imx-yocto-bsp-4-19-35-1-1-0/s
cd ~/projects/imx-yocto-bsp-4-19-35-1-1-0/sources
tar xzvf meta-custom.tgz
```

### Prepare an embedded Linux distribution for the MCIMX8M-EVK board

The bitbake target core-image-full-cmdline created for MCIMX8M-EVK contains a busybox implementation of the usual Unix command line tools. It assumes you will interact with the embedded system over the command line:

```
cd ~/projects/imx-yocto-bsp-4-19-35-1-1-0
DISTRO=fsl-imx-wayland MACHINE=imx8mqevk source fsl-setup-release.sh -b build-wayland-
↳ imx8mqevk
```

Now, before issuing the bitbake command, edit two configuration files.

First edit the build-wayland-imx8mqevk/conf/bblayers.conf so it contains a reference to the custom layer. Add the following line at the end of the file:

```
BBLAYERS += " ${BSPDIR}/sources/meta-custom"
```

The resulting build-wayland-imx8mqevk/conf/bblayers.conf file will look as follows:

```
LCONF_VERSION = "6"

BBPATH = "${TOPDIR}"
BSPDIR := "${@os.path.abspath(os.path.dirname(d.getVar('FILE', True)) + '/../..')}"
```

(continues on next page)

(continued from previous page)

```

BBFILES ?= ""
BBLAYERS = " \
 ${BSPDIR}/sources/poky/meta \
 ${BSPDIR}/sources/poky/meta-poky \
 \
 ${BSPDIR}/sources/meta-openembedded/meta-oe \
 ${BSPDIR}/sources/meta-openembedded/meta-multimedia \
 \
 ${BSPDIR}/sources/meta-freescale \
 ${BSPDIR}/sources/meta-freescale-3rdparty \
 ${BSPDIR}/sources/meta-freescale-distro \
"

i.MX Yocto Project Release layers
BBLAYERS += " ${BSPDIR}/sources/meta-fsl-bsp-release/imx/meta-bsp "
BBLAYERS += " ${BSPDIR}/sources/meta-fsl-bsp-release/imx/meta-sdk "
BBLAYERS += " ${BSPDIR}/sources/meta-fsl-bsp-release/imx/meta-ml "

BBLAYERS += "${BSPDIR}/sources/meta-browser"
BBLAYERS += "${BSPDIR}/sources/meta-rust"
BBLAYERS += "${BSPDIR}/sources/meta-openembedded/meta-gnome"
BBLAYERS += "${BSPDIR}/sources/meta-openembedded/meta-networking"
BBLAYERS += "${BSPDIR}/sources/meta-openembedded/meta-python"
BBLAYERS += "${BSPDIR}/sources/meta-openembedded/meta-filesystems"
BBLAYERS += "${BSPDIR}/sources/meta-qt5"

+SIMW
BBLAYERS += " ${BSPDIR}/sources/meta-custom"
-SIMW

```

Next edit the file `build-wayland-imx8mqevk/conf/local.conf` so it matches the following:

```

MACHINE ??= 'imx8mqevk'
DISTRO ?= 'fsl-imx-wayland'
PACKAGE_CLASSES ?= 'package_rpm'

EXTRA_IMAGE_FEATURES ?= "debug-tweaks"
+ SIMW: Extended EXTRA_IMAGE_FEATURES
EXTRA_IMAGE_FEATURES ?=
↪ "debug-tweaks dev-pkgs tools-debug tools-sdk tools-testapps package-management"

USER_CLASSES ?= "buildstats image-mklibs image-prelink"
PATCHRESOLVE = "noop"
BB_DISKMON_DIRS ??= "\
 STOPTASKS,${TMPDIR},1G,100K \
 STOPTASKS,${DL_DIR},1G,100K \
 STOPTASKS,${SSTATE_DIR},1G,100K \
 STOPTASKS,/tmp,100M,100K \
 ABORT,${TMPDIR},100M,1K \
 ABORT,${DL_DIR},100M,1K \
 ABORT,${SSTATE_DIR},100M,1K \
 ABORT,/tmp,10M,1K"
PACKAGECONFIG_append_pn-qemu-system-native = " sdl"
PACKAGECONFIG_append_pn-nativesdk-qemu = " sdl"
CONF_VERSION = "1"

```

(continues on next page)

(continued from previous page)

```
DL_DIR ?= "${BSPDIR}/downloads/"
ACCEPT_FSL_EULA = "1"

+SIMW
IMAGE_ROOTFS_EXTRA_SPACE = "640000"

IMAGE_INSTALL_append += " rng-tools openssl-bin"
IMAGE_INSTALL_append += " cmake curl git subversion "
If you want git-submodule
IMAGE_INSTALL_append += " git-perltools findutils "
IMAGE_INSTALL_append += "
↪ " python3-pip python3-click python3-cryptography python3-pycparser python3-cffi "
opcu
IMAGE_INSTALL_append += " python-logging "

IMAGE_INSTALL_append += " e2fsprogs-resize2fs func-timeout "
IMAGE_INSTALL_append += " i2c-tools "
-SIMW
```

The above local.conf file prepares an embedded Linux distribution with native development tools

After updating the file build-wayland-imx8mqevk/conf/local.conf issue the following command:

```
bitbake core-image-full-cmdline
```

### Note: Output Directory

The directory *build-wayland-imx8mqevk* will contain downloaded sources and build artefacts.

When the above bitbake command finished successfully a compressed sdcard image containing bootloader, filesystem and linux kernel is available under `~/projects/imx-yocto-bsp-4-19-35-1-1-0/build-wayland-imx8mqevk/tmp/deploy/images/imx8mqevk/core-image-full-cmdline-imx8mqevk.wic.bz2`.

`core-image-full-cmdline-imx8mqevk.wic.bz2` is a symbolic link to the actual file name - which has a timestamp as part of the filename e.g. `core-image-full-cmdline-imx8mqevk-20191204234929.rootfs.wic.bz2`. Copy the **unzipped** sdcard image to a microSD card either with the method described in the 'i.MX Yocto Project User's Guide' or on a Windows PC with e.g. the Win32DiskImager tool.

**Note:** The embedded linux system prepared has a user `root` that does not require a password. Please define a password at your earliest convenience.



## Create and install SDK and Root file system on Host

To populate SDK, run:

```
cd ~/projects/imx-yocto-bsp-4-19-35-1-1-0
MACHINE=imx8mqevk source setup-environment build-wayland-imx8mqevk
bitbake -c populate_sdk core-image-full-cmdline
```

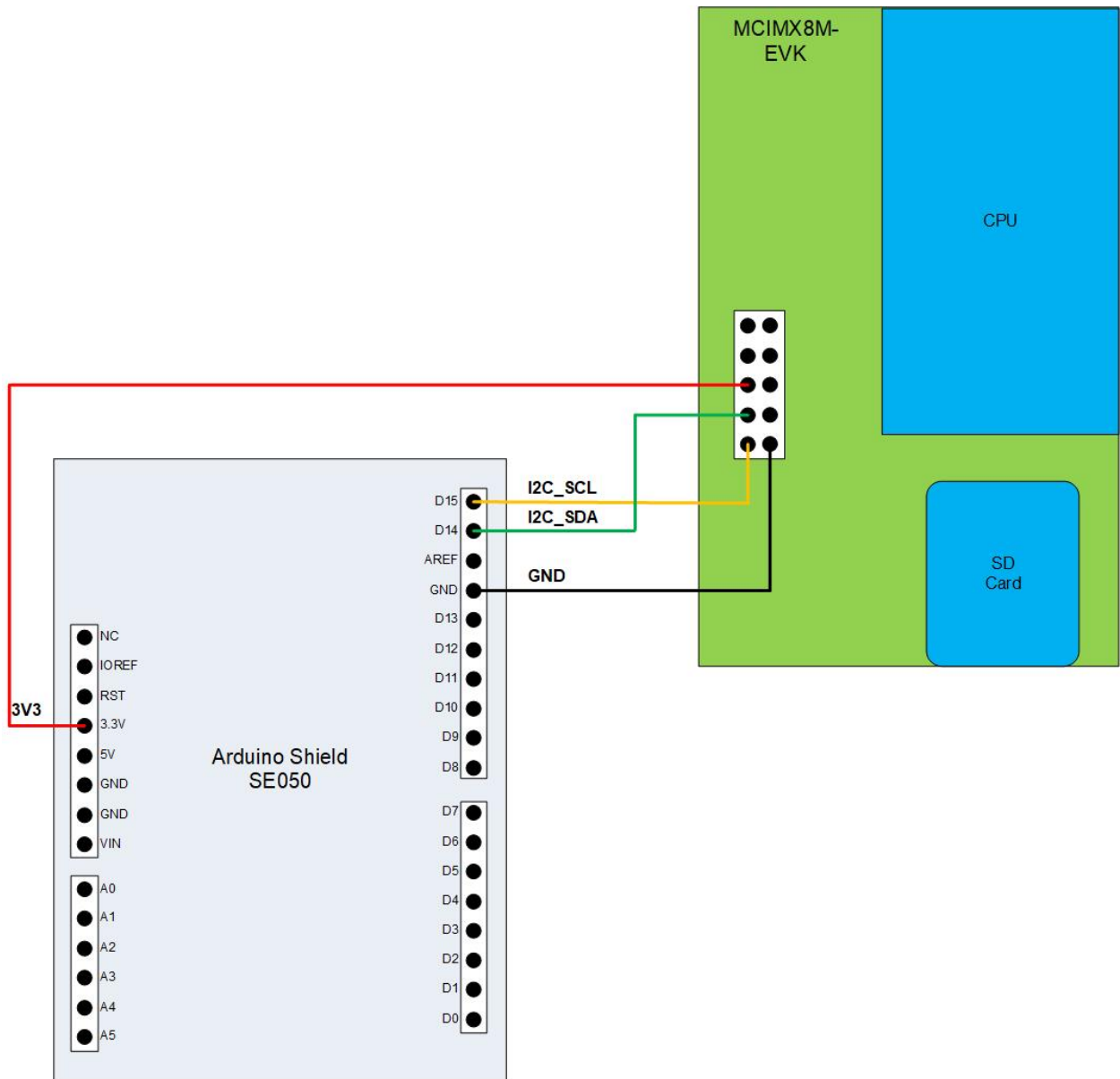
To install the SDK in the default location /opt/fsl-imx-wayland/4.19-warrior:

```
cd ~/projects/imx-yocto-bsp-4-19-35-1-1-0/build-wayland-imx8mqevk/tmp/deploy/sdk/
./fsl-imx-wayland-glibc-x86_64-core-image-full-cmdline-aarch64-toolchain-4.19-warrior.
↪ sh
```

You have now installed a set of cross-compilation tools for the MCIMX8M-EVK board on the Linux Host PC.

## Connecting the MCIMX8M-EVK to the SE050 Arduino shield

The MCIMX8M-EVK does not come with an Arduino connector, the following figure illustrates how to make the connection with jump wires.



The same information summarized in a table.

| Pin Function | OM-SE050ARD pins | MCIMX8M-EVK |
|--------------|------------------|-------------|
| I2C_SCL      | J2-10            | J801-I2C-1  |
| I2C_SDA      | J2-9             | J801-I2C-3  |
| GND          | J2-7             | J801-I2C-2  |
| 3V3          | J8-4             | J801-I2C-5  |

## 9.5.2 Setup i.MX6UL - MCIMX6UL-EVK

This section explains how to create an SD card image (including native compilation tools) and a cross-compilation environment for the MCIMX6UL-EVK.

**Note:** *Setup i.MX 8MQuad - MCIMX8M-EVK* is the current default version of the board used with the Plug & Trust MW.

### Downloading and Installing Yocto for MCIMX6UL-EVK

Please consult first the detailed information that can be found on <https://www.nxp.com>. Download the L4.19.35\_1.1.0\_LINUX\_DOCS documentation package (available on <https://www.nxp.com/design/i-mx-developer-resources/i-mx-software-and-development-tools:IMX-SW> under the header **Overview - Documentation - Linux 4.19.35\_1.1.0 Documentation** and take the **i.MX Yocto Project User's Guide** as a starting point.

Yocto must be installed on a Linux PC (consult the documentation for the Linux distributions officially supported), it's possible to use a Virtual PC environment as e.g. Virtual Box. The Linux PC must have access to the internet to retrieve additional packages, tools and sources.

Having set-up all tools and packages required by Yocto, initialize a local git repository as follows:

```
mkdir -p ~/projects/imx-yocto-bsp-4-19-35-1-1-0
cd ~/projects/imx-yocto-bsp-4-19-35-1-1-0
repo init -u https://source.codeaurora.org/external/imx/imx-manifest -b imx-linux-
↳warrior -m imx-4.19.35-1.1.0.xml
repo sync
```

Add a custom Yocto layer. This custom Yocto layer is part of the Plug&Trust SW distribution (simw\_top/scripts/yocto/layers/meta-custom.tgz) and must be copied and unpacked into the sources directory created above. Note: this custom layer adds a Python package (func-timeout):

```
cp_
↳<PlugTrust>/simw_top/scripts/yocto/layers/meta-custom.tgz ~/projects/imx-yocto-bsp-4-19-35-1-1-0/s
cd ~/projects/imx-yocto-bsp-4-19-35-1-1-0/sources
tar xzvf meta-custom.tgz
```

### Prepare an embedded Linux distribution for the MCIMX6UL-EVK board

The bitbake target core-image-full-cmdline created for MCIMX6UL-EVK contains a busybox implementation of the usual Unix command line tools. It assumes you will interact with the embedded system over the command line:

```
cd ~/projects/imx-yocto-bsp-4-19-35-1-1-0
DISTRO=fsl-imx-fb MACHINE=imx6ulevk source fsl-setup-release.sh -b build-fb-6ul
```

Now, before issuing the bitbake command, edit two configuration files.

First edit the build-fb-6ul/conf/bblayers.conf so it contains a reference to the custom layer. Add the following line at the end of the file:

```
BBLAYERS += " ${BSPDIR}/sources/meta-custom"
```

The resulting build-fb-6ul/conf/bblayers.conf file will look as follows:

```
LCONF_VERSION = "7"

BBPATH = "${TOPDIR}"
BSPDIR := "${@os.path.abspath(os.path.dirname(d.getVar('FILE', True)) + '/../..')}"

BBFILES ?= ""
BBLAYERS = " \
 ${BSPDIR}/sources/poky/meta \
 ${BSPDIR}/sources/poky/meta-poky \
 \
 ${BSPDIR}/sources/meta-openembedded/meta-oe \
 ${BSPDIR}/sources/meta-openembedded/meta-multimedia \
 \
 ${BSPDIR}/sources/meta-freescale \
 ${BSPDIR}/sources/meta-freescale-3rdparty \
 ${BSPDIR}/sources/meta-freescale-distro \
"

i.MX Yocto Project Release layers
BBLAYERS += " ${BSPDIR}/sources/meta-fsl-bsp-release/imx/meta-bsp "
BBLAYERS += " ${BSPDIR}/sources/meta-fsl-bsp-release/imx/meta-sdk "
BBLAYERS += " ${BSPDIR}/sources/meta-fsl-bsp-release/imx/meta-ml "

BBLAYERS += "${BSPDIR}/sources/meta-browser"
BBLAYERS += "${BSPDIR}/sources/meta-rust"
BBLAYERS += "${BSPDIR}/sources/meta-openembedded/meta-gnome"
BBLAYERS += "${BSPDIR}/sources/meta-openembedded/meta-networking"
BBLAYERS += "${BSPDIR}/sources/meta-openembedded/meta-python"
BBLAYERS += "${BSPDIR}/sources/meta-openembedded/meta-filesystems"
BBLAYERS += "${BSPDIR}/sources/meta-qt5"
BBLAYERS += " ${BSPDIR}/sources/meta-custom "
```

Next edit the file `build-fb-6ul/conf/local.conf` so it matches the following:

```
MACHINE ??= 'imx6ulevk'
DISTRO ?= 'fsl-imx-fb'
PACKAGE_CLASSES ?= "package_rpm"
EXTRA_IMAGE_FEATURES ?=
 ↪ "debug-tweaks dev-pkgs tools-debug tools-sdk tools-testapps package-management"
USER_CLASSES ?= "buildstats image-mklibs image-prelink"
PATCHRESOLVE = "noop"
BB_DISKMON_DIRS ??= " \
 STOPTASKS, ${TMPDIR}, 1G, 100K \
 STOPTASKS, ${DL_DIR}, 1G, 100K \
 STOPTASKS, ${SSTATE_DIR}, 1G, 100K \
 STOPTASKS, /tmp, 100M, 100K \
 ABORT, ${TMPDIR}, 100M, 1K \
 ABORT, ${DL_DIR}, 100M, 1K \
 ABORT, ${SSTATE_DIR}, 100M, 1K \
 ABORT, /tmp, 10M, 1K"
PACKAGECONFIG_append_pn-qemu-system-native = " sdl"
PACKAGECONFIG_append_pn-nativesdk-qemu = " sdl"
CONF_VERSION = "1"
IMAGE_ROOTFS_EXTRA_SPACE = "640000"

DL_DIR ?= "${BSPDIR}/downloads/"
ACCEPT_FSL_EULA = "1"
```

(continues on next page)

(continued from previous page)

```

IMAGE_INSTALL_append += " rng-tools openssl-bin"
IMAGE_INSTALL_append += " cmake curl git subversion"
IMAGE_INSTALL_append +=
→ " python3-pip python3-click python3-cryptography python3-pycparser python3-cffi"
IMAGE_INSTALL_append += " e2fsprogs-resize2fs func-timeout i2c-tools"

```

The above local.conf file prepares an embedded Linux distribution with native development tools

After updating the file build-fb-6ul/conf/local.conf issue the following command:

```
bitbake core-image-full-cmdline
```

---

#### Note: Output Directory

The directory *build-fb-6ul* will contain downloaded sources and build artefacts.

---

When the above bitbake command finished successfully an sdcard image containing bootloader, filesystem and linux kernel is available under `~/projects/imx-yocto-bsp-4-14-98/build-fb-6ul/tmp/ deploy/images/imx6ulevk/core-image-full-cmdline-imx6ulevk.sdcard`.

`core-image-full-cmdline-imx6ulevk.sdcard` is a symbolic link to the actual file name - which has a timestamp as part of the filename e.g. `core-image-full-cmdline-imx6ulevk-20170825222257.rootfs.sdcard`. Copy this sdcard image to a microSD card either with the method described in the 'i.MX Yocto Project User's Guide' or on a Windows PC with e.g. the Win32DiskImager tool. \*/

When the above bitbake command finished successfully a compressed sdcard image containing bootloader, filesystem and linux kernel is available under `~/projects/imx-yocto-bsp-4-19-35-1-1-0/build-fb-6ul/tmp/ deploy/images/imx6ulevk/core-image-full-cmdline-imx6ulevk.wic.bz2`.

`core-image-full-cmdline-imx6ulevk.wic.bz2` is a symbolic link to the actual file name - which has a timestamp as part of the filename e.g. `core-image-full-cmdline-imx6ulevk-20191204234929.rootfs.wic.bz2`. Copy the **unzipped** sdcard image to a microSD card either with the method described in the 'i.MX Yocto Project User's Guide' or on a Windows PC with e.g. the Win32DiskImager tool.

---

**Note:** The embedded linux system prepared has a user `root` that does not require a password. Please define a password at your earliest convenience.

---

### Create and install SDK and Root file system on Host

To populate SDK, run:

```

cd ~/projects/imx-yocto-bsp-4-19-35-1-1-0
MACHINE=imx6ulevk source setup-environment build-fb-6ul
bitbake -c populate_sdk core-image-full-cmdline

```

To install the SDK in the default location `/opt/fsl-imx-fb/4.9.88-2.0.0`:

```

cd ~/projects/imx-yocto-bsp-4-19-35-1-1-0/build-fb-6ul/tmp/deploy/sdk/
./fsl-imx-fb-glibc-x86_64-core-image-full-cmdline-cortexa7hf-neon-toolchain-4.9.88-2.
→ 0.0.sh

```

You have now installed a set of cross-compilation tools for the MCIMX6UL-EVK board on the Linux Host PC.

## Create and install SDK and Root file system on Host

To populate SDK, run:

```
cd ~/projects/imx-yocto-bsp-4-19-35-1-1-0
MACHINE=imx6ulevk source setup-environment build-fb-6ul
bitbake -c populate_sdk core-image-full-cmdline
```

To install the SDK in the default location /opt/fsl-imx-fb/4.19-warrior:

```
cd ~/projects/imx-yocto-bsp-4-19-35-1-1-0/build-fb-6ul/tmp/deploy/sdk/
./fsl-imx-fb-glibc-x86_64-core-image-full-cmdline-aarch64-toolchain-4.19-warrior.sh
```

You have now installed a set of cross-compilation tools for the MCIMX6UL-EVK board on the Linux Host PC.

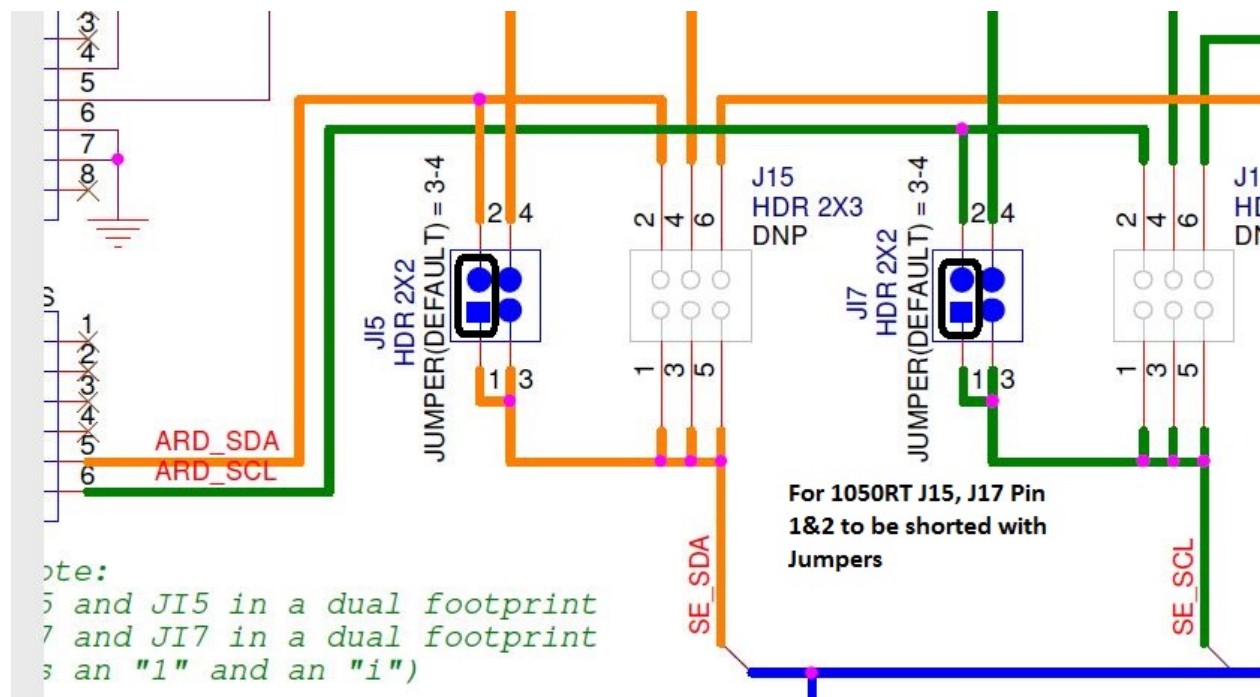
### 9.5.3 Freedom K64F

See Section 4.2 — *Import MCUXPresso projects from SDK* and Section 4.3 — *Freedom K64F Build (CMake - Advanced)*

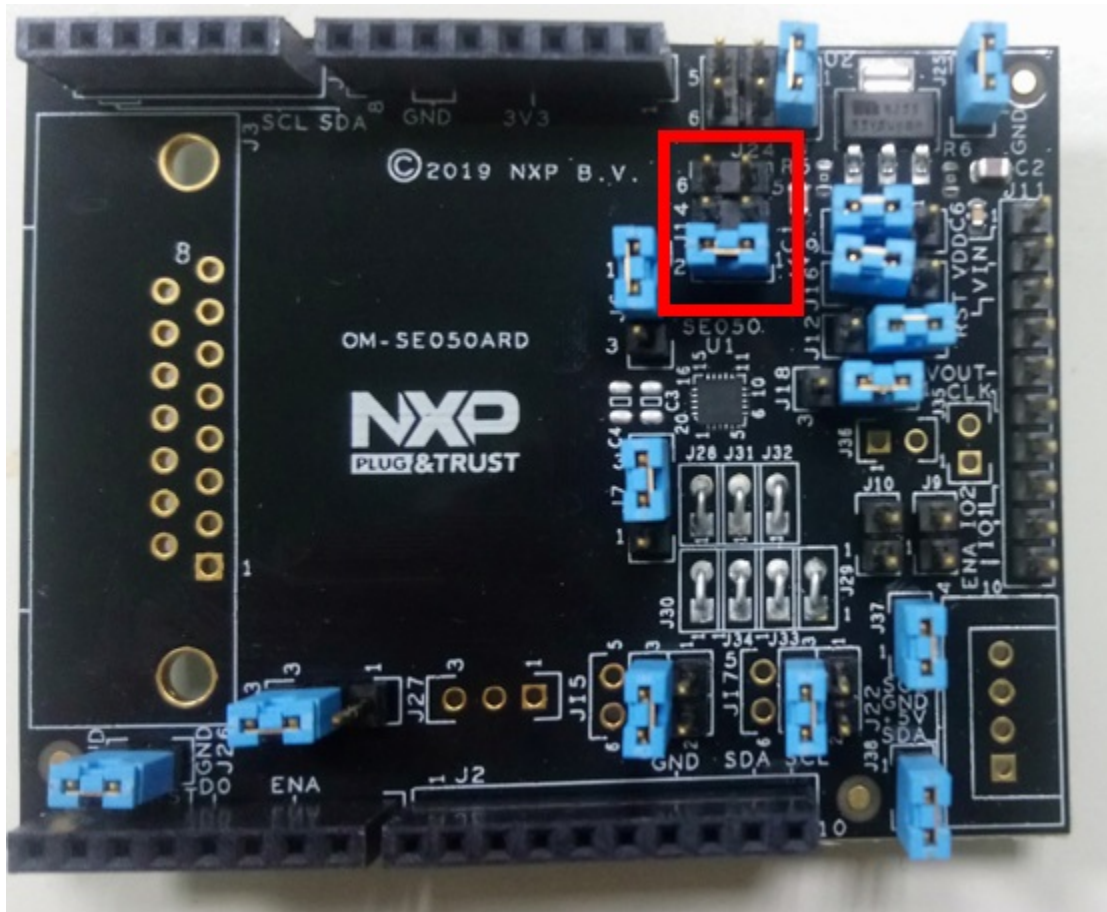
### 9.5.4 Connecting SE050 Arduino shield to imxRT1050

SDA and SCL are connected differently for RT1050 as compared to FRDM-K64F

- ARD\_SDA is Required on pin 5 of SE050 Arduino Shield
- ARD\_SCL is Required on pin 6 of SE050 Arduino Shield



Connect SE050 to imxRT1050 Arduino stackable headers and change jumper J14 as:

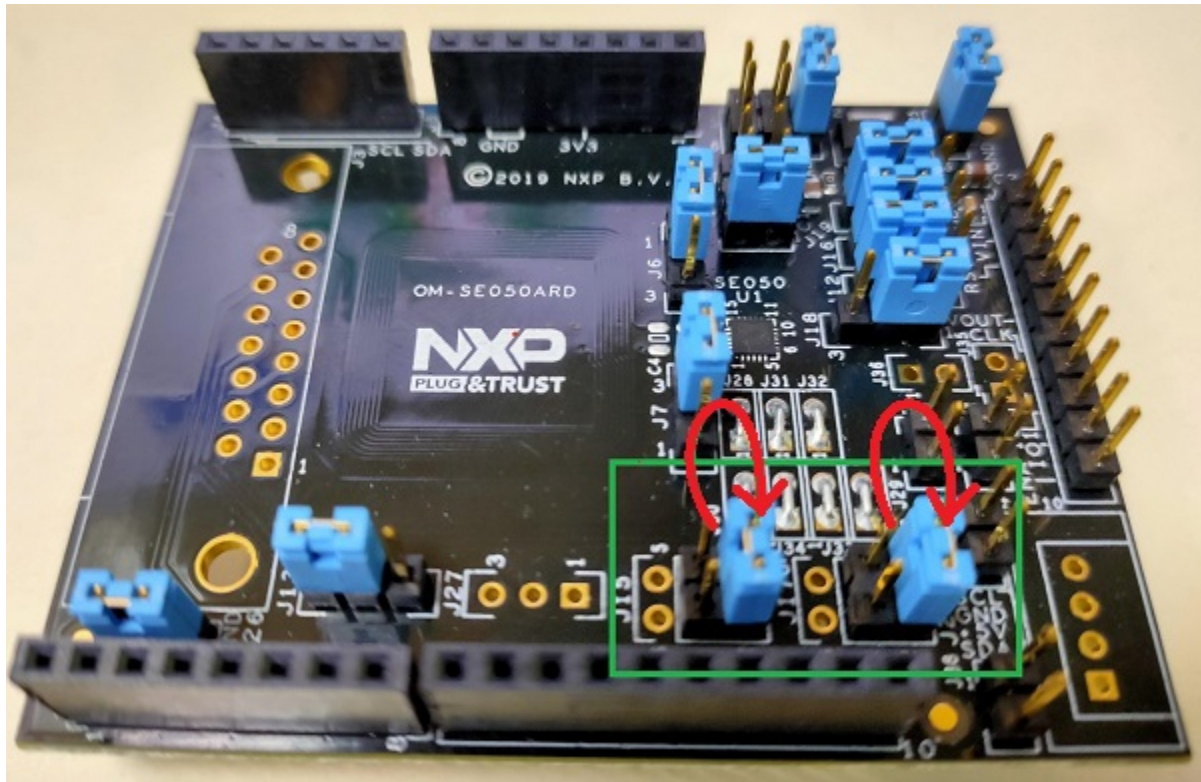


This connects SE\_VDD directly to 3V3 and bypasses enable signal. This is required because enable pin on imxRT1050 coincides with RC-663 nRST pin so we cannot use SE\_EN signal to drive SE\_VDD.

#### Jumper Settings for SE050 Arduino Shield and 1050RT

- SDA J15 Pin 1 & 2 connected using jumper
- SCL J17 Pin 1 & 2 connected using jumper



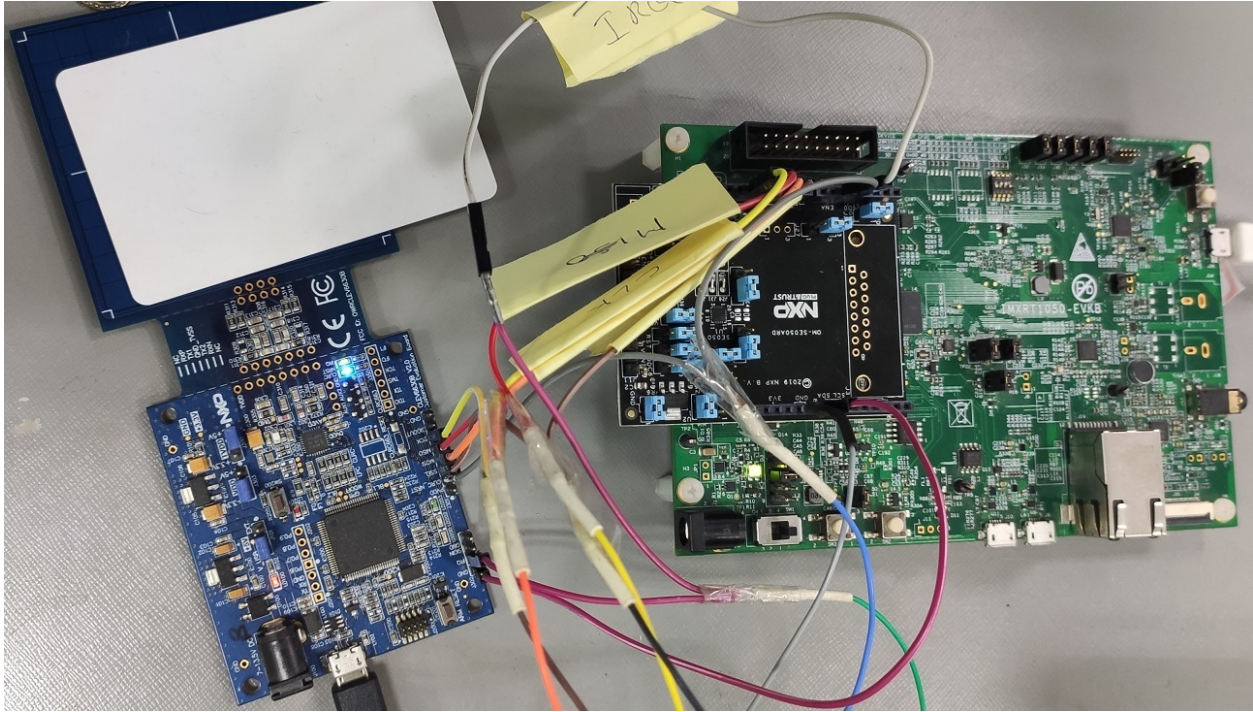


### 9.5.5 Connecting RC663 to imxRT1050

| Pin Function | MIMXRT1050 pins | CLEV6630B  |
|--------------|-----------------|------------|
| MOSI         | J24-4           | MOSI       |
| MISO         | J24-5           | MISO       |
| SPI SCK      | J24-6           | SCK        |
| SPI CSEL1    | J24-3           | SSEL       |
| RESET        | J22-6           | CLRCL_NRST |
| IRQ          | J22-5           | IRQ        |
| GND          | J25             | GND        |

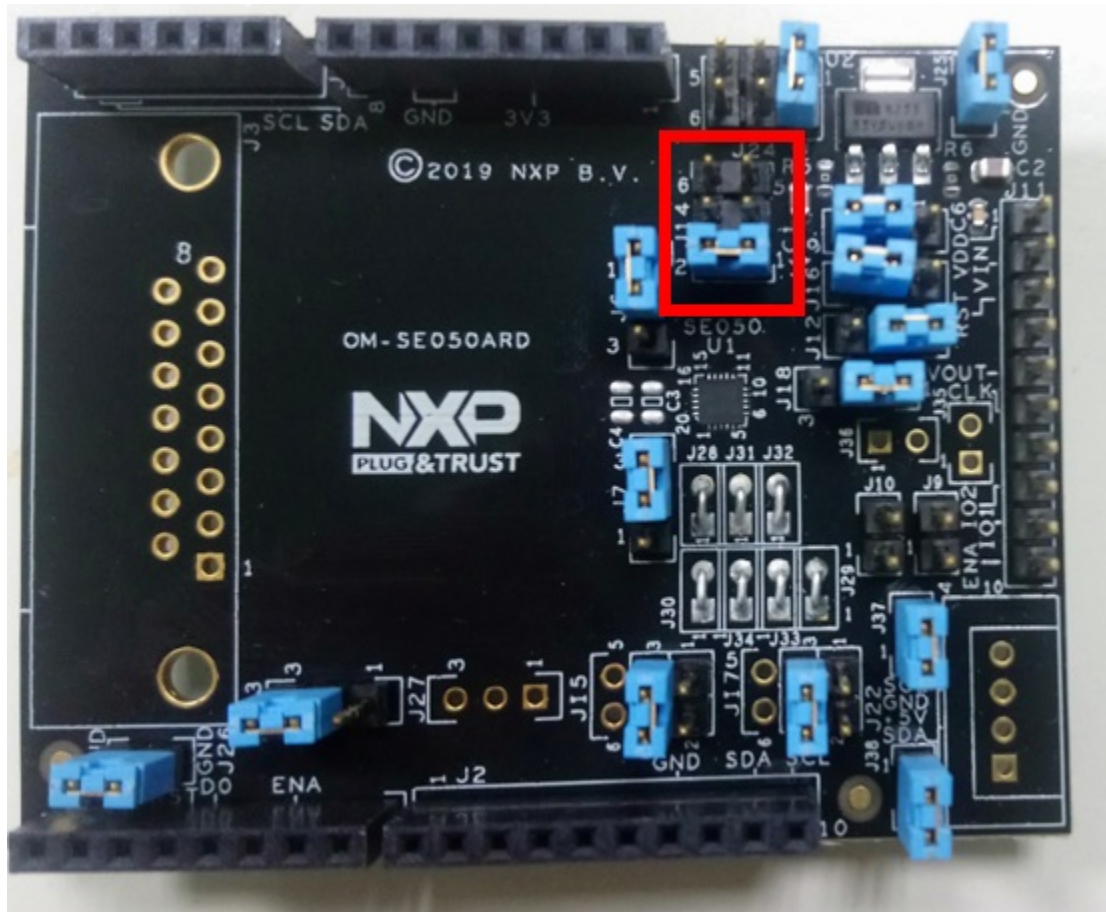
**Note:** in a new MIMXRT1050 Board, Resistors R278, R279, R280, R281 are DNP. So they either have to be shorted or a 0 OHM resistor has to be populated.





### 9.5.6 Connecting SE050 Arduino shield to LPC555

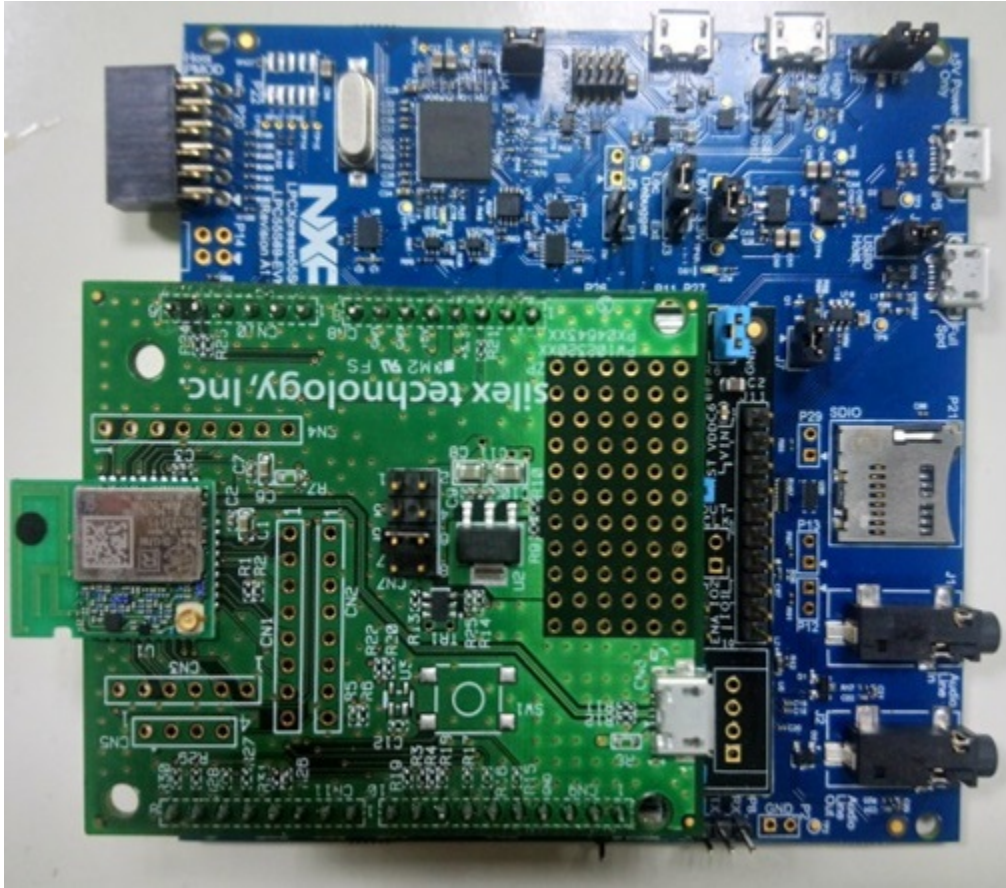
Connect SE050 to LPC555 Arduino stackable headers and change jumper J14 as:



This connects SE\_VDD directly to 3V3 and bypasses enable signal. This is required because enable pin on LPC55S coincides with Sillex-2401 SPI pins so we cannot use SE\_EN signal to drive SE\_VDD.

### 9.5.7 Connecting WiFi shield Sillex-2401 to LPC55S

Connect Sillex-2401 WiFi shield with SE050 as:



### 9.5.8 Connecting CLEV6630B to FRDM-K64F or LPC55S69

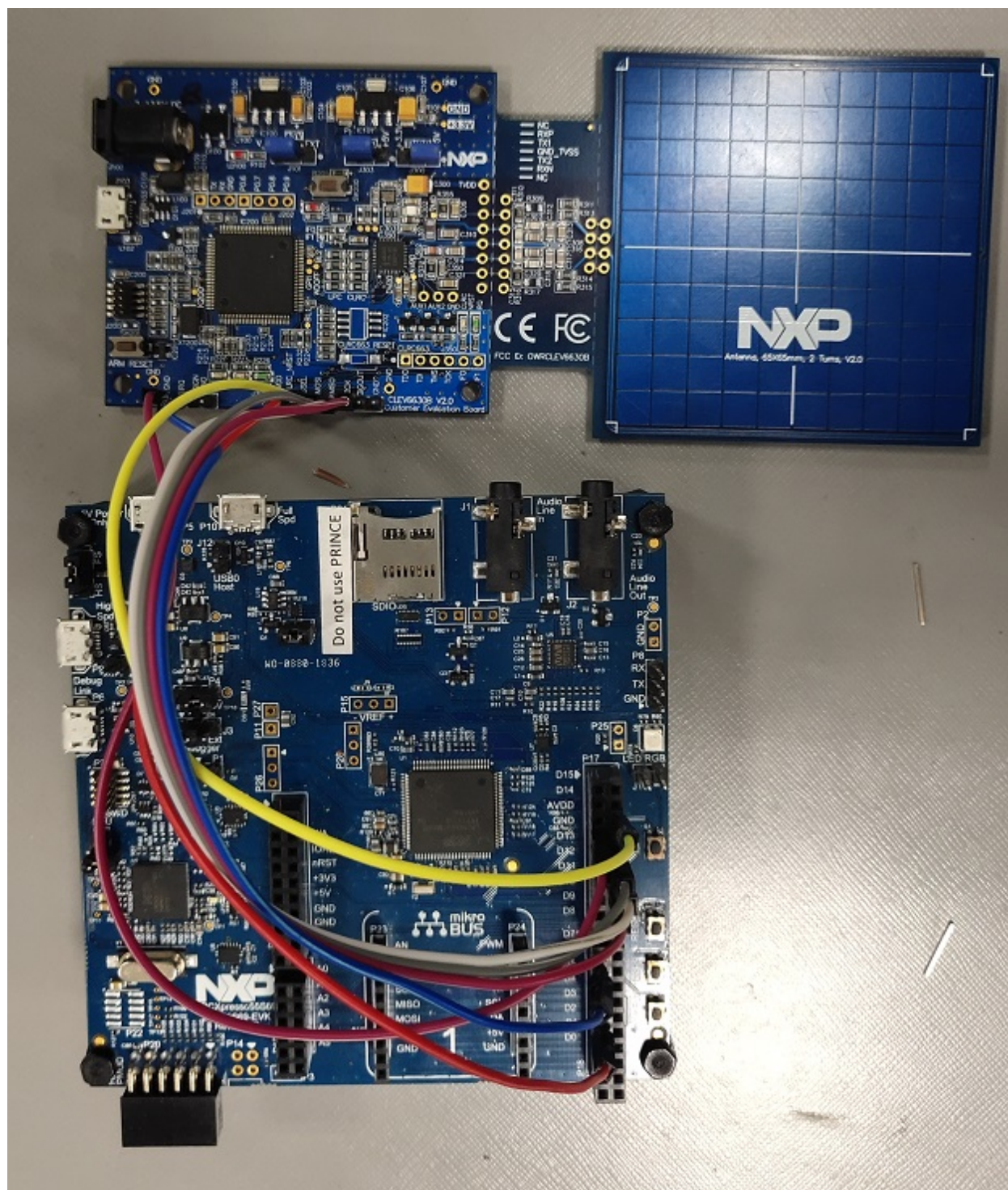
This is required for Setting up the NFC MIFARE DESFire EV2 examples.

Setup the hardware as per the connection table below.

#### Connections between LPC55S69 Board and CLEV6630B

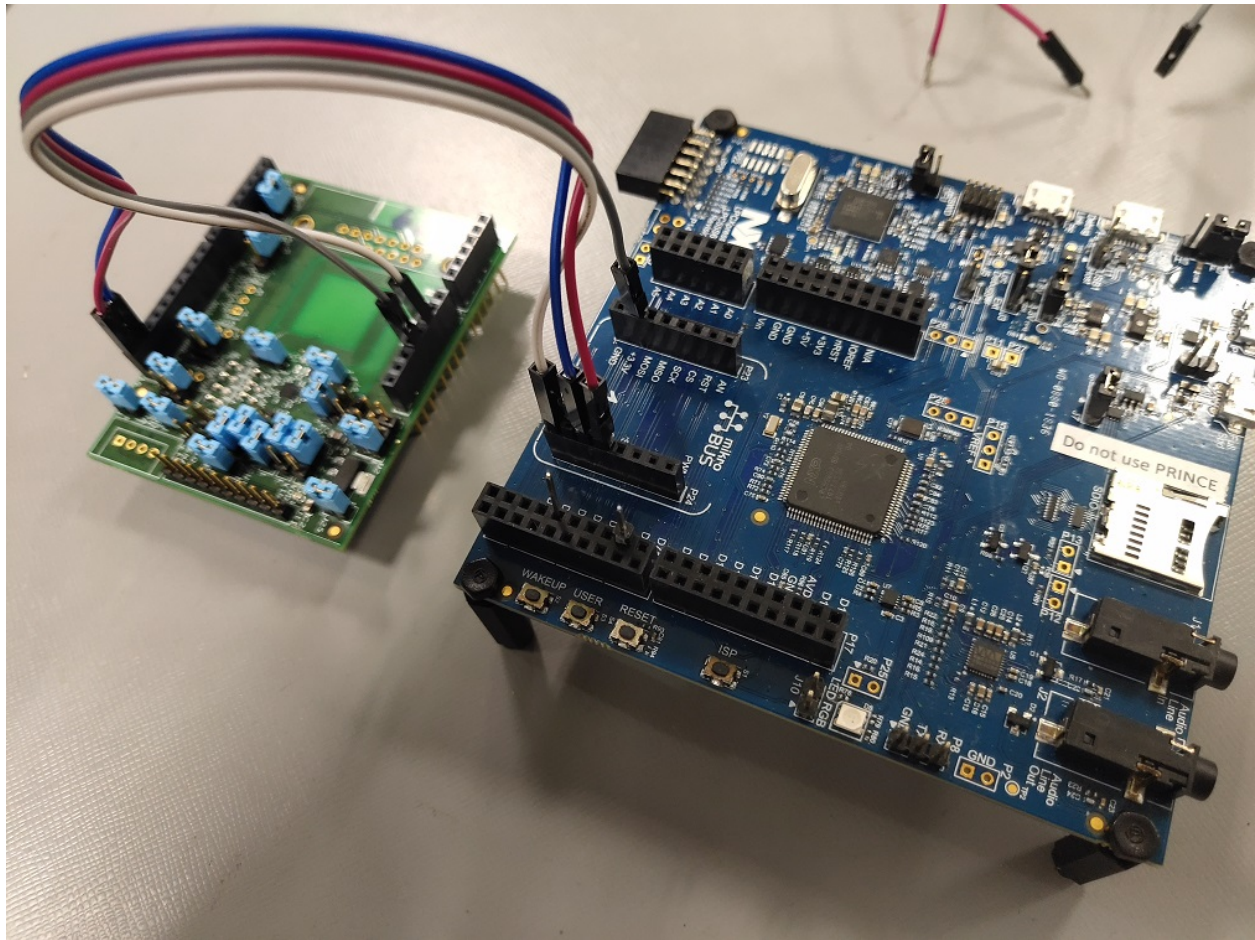
| Pin Function | LPC55S69 (Jumper # - Pin #) | Port    | CLEV6630B  |
|--------------|-----------------------------|---------|------------|
| MOSI         | P17-10                      | PIO0_20 | MOSI       |
| MISO         | P17-12                      | PIO0_19 | MISO       |
| SPI SCK      | P17-14                      | PIO0_21 | SCK        |
| SPI CSEL     | P17-6                       | PIO1_11 | SSEL       |
| RESET        | P18-11                      | PIO0_15 | CLRCL_NRST |
| IRQ          | P18-3                       | PIO1_10 | IRQ        |
| GND          | P17-7                       |         | GND        |





## connections between LPC55S69 and OM-SE050ARD

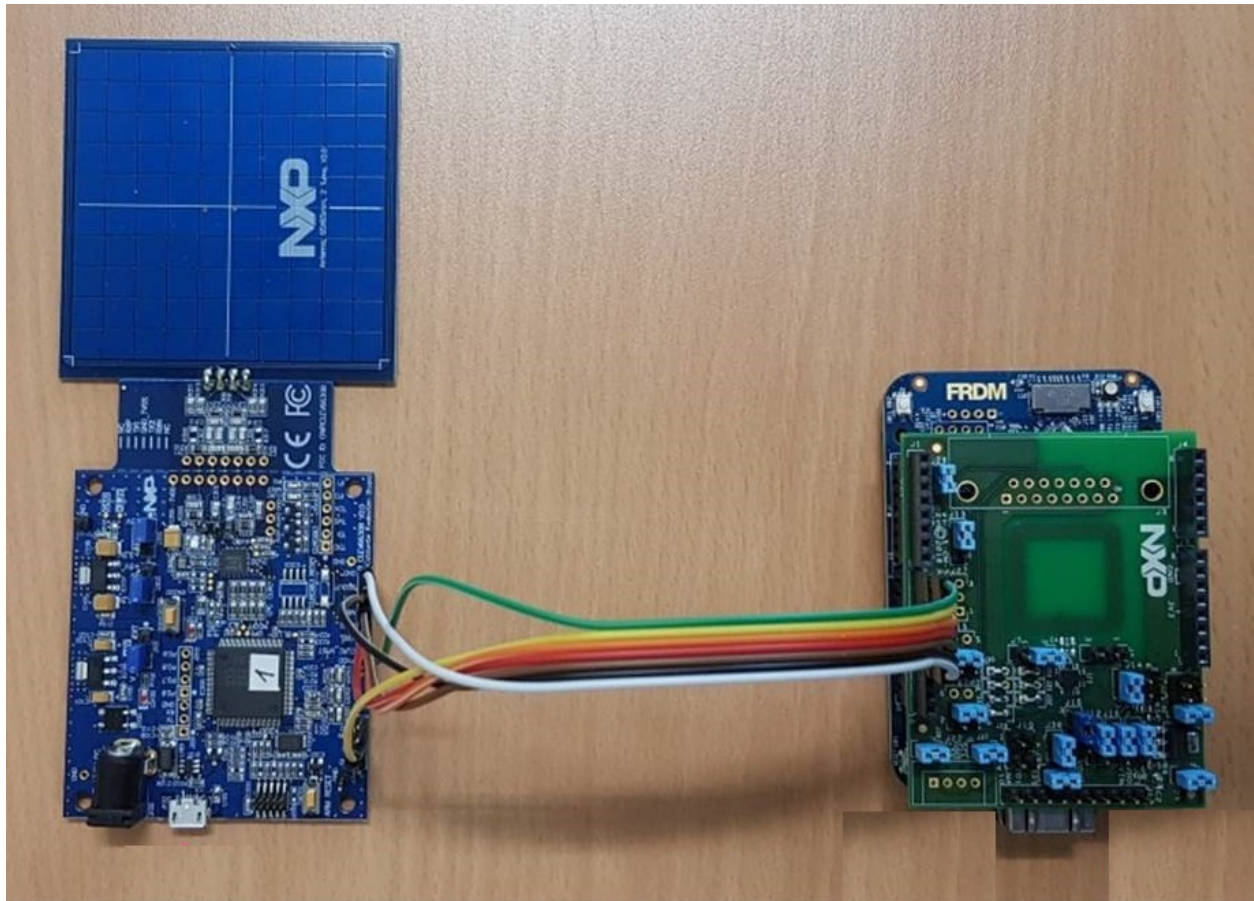
| OM-SE050ARD    | LPC55S69 (Jumper # - Pin #) | Port    | Function Name   |
|----------------|-----------------------------|---------|-----------------|
| SE_SDA (J22-1) | P24-6 also P17-3            | PIO1_21 | FC4_I2C_SDA_ARD |
| SE_SCL (J22-4) | P24-5 also P17-1            | PIO1_20 | FC4_I2C_SCL_ARD |
| +5V_PC (J22-2) |                             |         | VDD_TARGET      |
| GND (J22-3)    |                             |         | GND             |



## Connections between FRDM-K64F Board and CLEV6630B

| Pin Function | K64F pins    | CLEV6630B  |
|--------------|--------------|------------|
| MOSI         | J2-8-PDT2    | MOSI       |
| MISO         | J2-10-PDT3   | MISO       |
| SPI SCK      | J2-12-PDT1   | SCK        |
| SPI CSEL1    | J2-6-PDT0    | SSEL       |
| RESET        | J2-2-PTA0    | CLRCL_NRST |
| IRQ          | J2-4-PTC4    | IRQ        |
| GND          | J2-14        | GND        |
| 3V3          | J3-4 or J3-8 |            |





Also, A firmware needs to be flashed into LPC1769 of CLEV6630B to ensure that it does not drive any of the RC663 Signals. A firmware binary is stored at `demos\nxpnfrdlib\LPC_RC663`

## 9.5.9 Android

### Supported Features

#### Generate / Import:

- ECC:
  - 224-bit
  - 256-bit
  - 384-bit
  - 521-bit
- RSA:
  - 1024-bit
  - 2048-bit
  - 3072-bit
  - 4096-bit
- AES:

- 128-bit
- 192-bit
- 256-bit
- **HMAC**
  - 64-bit to 512-bit

**Export:**

- **ECC:**
  - 224-bit
  - 256-bit
  - 384-bit
  - 521-bit
- **RSA:**
  - 1024-bit
  - 2048-bit
  - 3072-bit
  - 4096-bit

**RNG:**

- Get Random number

**Sign / Verify:**

- **Sign / Verify with ECC**
  - **Supported digests:**
    - DIGEST:SHA1
    - DIGEST:SHA224
    - DIGEST:SHA256
    - DIGEST:SHA384
    - DIGEST:SHA512
- **Sign / Verify with RSA**
  - **Supported paddings:**
    - PADDING:NONE
    - PADDING:PKCS1\_V1.5 (SHA1, SHA-224, SHA-256, SHA-384, SHA-512)
    - PADDING:PSS (SHA1, SHA-224, SHA-256, SHA-384, SHA-512)
- **Sign / Verify with HMAC**
  - **Supported digests:**
    - DIGEST:SHA1
    - DIGEST:SHA256
    - DIGEST:SHA384

- DIGEST:SHA512

#### Encryption / Decryption:

- **Encrypt / Decrypt with RSA**
  - **Supported paddings:**
    - PADDING:NONE
    - PADDING:PKCS1\_V1.5
    - PADDING:OAEP (SHA1)
- **Encrypt / Decrypt with AES**
  - **Supported block modes:**
    - ECB (PADDING:NONE)
    - CBC (PADDING:NONE)
    - CTR (PADDING:NONE)

#### Delete:

- Single key
- All keys

#### Attestation:

- Key attestation

## AOSP build Environment Setup

### AOSP build Environment for Hikey960

To setup Android build environment for Hikey960 board please follow steps below:

- 1) The build setup file structure should be as below:

```
<ROOT-DIR>
|
|----- android-root/
|
|----- simw-top/
```

- 2) Downloading and building AOSP source code (refer <https://source.android.com/setup/build/devices>).

In the steps below, android-root means `$ROOT_DIR/android-root/`.

For simplicity of scripts, it is assumed that `ROOT_DIR` variable is set like as below:

```
ROOT_DIR=/opt/_ddm/aospbld
```

- 3) Setup REPO Tool:

```
mkdir ~/bin
PATH=~/bin:$PATH
curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
chmod a+x ~/bin/repo
```

- 4) Download source code using REPO tool:



```

ROOT_DIR=/opt/_ddm/aospbld # For example

mkdir -p ${ROOT_DIR}/android-root
cd ${ROOT_DIR}/android-root

repo init -u https://android.googlesource.com/platform/manifest -b android-p-
↳ preview-2
repo sync -j$(nproc)

```

All scripts from here on assume ROOT\_DIR is set.

5) Apply patches from android.googlesource.com:

```

cd ${ROOT_DIR}/android-root/prebuilts/tools
git fetch https://android.googlesource.com/platform/prebuilts/tools refs/changes/
↳ 02/682002/1 && git cherry-pick FETCH_HEAD

cd ${ROOT_DIR}/android-root/external/e2fsprogs/
git fetch https://android.googlesource.com/platform/external/e2fsprogs refs/
↳ changes/05/683305/1 && git cherry-pick FETCH_HEAD

cd ${ROOT_DIR}/android-root/external/f2fs-tools
git fetch https://android.googlesource.com/platform/external/f2fs-tools refs/
↳ changes/06/683306/1 && git cherry-pick FETCH_HEAD

```

6) Apply patches from host library.

These patches are for Android Keymaster 3.0 Board init

Scripts to apply the patches:

```

cp ${ROOT_DIR}/simw-top/akm/src/Board_init/keymaster_sepolicy.patch ${ROOT_DIR}/
↳ android-root/system/sepolicy/
cd ${ROOT_DIR}/android-root/system/sepolicy/
patch -p1 < keymaster_sepolicy.patch

cp ${ROOT_DIR}/simw-top/akm/src/Board_init/init_rc_file.patch ${ROOT_DIR}/android-
↳ root/system/core/
cd ${ROOT_DIR}/android-root/system/core/
patch -p1 < init_rc_file.patch

```

- init\_rc\_file.patch is to update system ownership of I2C module and to create /data/vendor/SE05x secure directory.
- keymaster\_sepolicy.patch is to update SE050 Keymaster HAL policy for accessing I2C device for communication with SE050 and /data/vendor/SE05x secure directory for storing Platform SCP03 keys.

7) Follow below instructions to build source code for hikey960:

```

cd ${ROOT_DIR}/android-root
export ANDROID_ROOT=$(pwd)
source build/envsetup.sh
lunch hikey960-userdebug
make -j $(nproc)

```

**Note:** Based on CPU core, build will take 1-4 hrs.

- 8) Installing images.

Follow <https://source.android.com/setup/build/devices#960fastboot>

- 9) Flashing images. Follow <https://source.android.com/setup/build/devices#960images>

---

**Note:** “fastboot” and “adb” are required for flashing images.

---

- 10) If modifications are required to hikey kernel (e.g. add/remove device driver), please refer to <https://source.android.com/setup/build/devices#960kernel> for building hikey kernel and follow the instruction given on link to create new bootimage image.

## AOSP build environment for iMX8M (coming soon)

To setup Android build environment for iMX8M board please follow below steps

1. The build setup file structure should be like (your\${ROOT\_DIR} dir):

```
<ROOT-DIR>
|
|----- android-root/
|
|----- simw-top/
```

- 1) Downloading and building AOSP source code (refer section 3.2.3 : Build your own Android BSP Image from [https://www.nxp.com/support/developer-resources/run-time-software/i.mx-developer-resources/evaluation-kit-for-the-i.mx-8m-applications-processor:MCIMX8M-EVK?tab=In-Depth\\_Tab](https://www.nxp.com/support/developer-resources/run-time-software/i.mx-developer-resources/evaluation-kit-for-the-i.mx-8m-applications-processor:MCIMX8M-EVK?tab=In-Depth_Tab)). In the steps below, android-root means \$ROOT\_DIR/android-root/.

- 2) Setup REPO Tool:

```
mkdir -p ${ROOT_DIR}/android-root
cd ${ROOT_DIR}/android-root

mkdir ~/bin
PATH=~/bin:$PATH
curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
chmod a+x ~/bin/repo
```

- 3) Download source code using REPO tool

Get the Android source code from Google repo using the manifest and script provided inside the imx-o8.1.0\_1.3.0\_8m.tar.gz (Download package from [https://www.nxp.com/support/developer-resources/run-time-software/i.mx-developer-resources/evaluation-kit-for-the-i.mx-8m-applications-processor:MCIMX8M-EVK?tab=Design\\_Tools\\_Tab](https://www.nxp.com/support/developer-resources/run-time-software/i.mx-developer-resources/evaluation-kit-for-the-i.mx-8m-applications-processor:MCIMX8M-EVK?tab=Design_Tools_Tab) ).

```
source ~/imx-o8.0.0_1.3.0_8mq-prc/imx_android_setup.sh

By default, the imx_android_setup.sh script will create the source code build_
↪environment
in the folder ~/android_build

${MY_ANDROID} will be referred as the i.MX Android source code root directory in_
↪all i.MX
Andorid release documentation.

export MY_ANDROID=~/android_build
```

## 4) Building Android images:

```
cd ${MY_ANDROID}
export ANDROID_ROOT=$(pwd)
source build/envsetup.sh
lunch evk_8mq-userdebug
make -j$(nproc) 2>&1 | tee build-log.txt
```

**Note:** Based on CPU core, build will take 1-4 hrs.

## 5) Flashing newly generate images.

- a) The board images can be flashed to the target board by using the MFGTool. The release package includes MFGTool for i.MX 8MQuad EVK in android\_08.0.0\_1.3.0\_8M-PRC\_tools.tar.gz. The MFGTool is mfgtools-mx8mq-beta.zip.
- b) Unzip the mfgtools-mx8mq-beta.zip file to a selected location. The directory is named MFGTool-Dir.

#) Copy following files from \$ROOT\_DIR/android-root/out/target/product/evk\_8mq to your MFGTool-Dir/Profiles/Linux/OS Firmware/files/ android/evk directory.

```
u-boot-imx8mq.img
partition-table.img
boot-imx8mq.img
vbmeta-imx8mq.img
system.img
vendor.img.
```

## SE050 based Android Keymaster

## CMAKE based build system

1. Download Android NDK from <https://developer.android.com/ndk/downloads/> and store it in /usr/local/ eg. /usr/local/android-ndk-r18b-linux-x86\_64

```
cd /usr/local/
wget http://dl.google.com/android/repository/android-ndk-r18b-linux-x86_64.zip
unzip -d android-ndk-r18b-linux-x86_64 android-ndk-r18b-linux-x86_64.zip
```

2. Once you are able to bring-up Android build environment for hikey960 follow below steps to build SE050 based android keymaster:

```
cd ${ROOT_DIR}/android-root
export ANDROID_ROOT=$(pwd)
cd ${ROOT_DIR}/simw-top/scripts/android/cmake_based
source board_config.sh hikey960
./setup_script.sh
```

After successful execution you will be able to locate <simw-top\_build> directory parallel to simw-top directory and simw-akm directory in \$ROOT\_DIR/android-root/system/keymaster

**Note:** If the patches are already applied, then instead of calling setup\_script.sh, call build\_script.sh

3. A batch script `keymaster_flash.bat` will be copied to `$ROOT_DIR/android-root/out/target/product/<BOARD_NAME>`. Execute the batch script to push all the necessary files onto the target board.

### AOSP based build system

1. Setup `simw-top` inside `$ROOT_DIR/android-root/vendor/nxp`. If `vendor/nxp` does not exist inside `$ROOT_DIR/android-root` then create the same.
2. Follow below steps to build SE050 based android keymaster.:

```
cd ${ROOT_DIR}/android-root
cp vendor/nxp/simw-top/akm/src/interface_keymaster/patch/aosp/interface_
↪keymaster3.0.patch hardware/interfaces/
cd hardware/interfaces/
patch -p1 < interface_keymaster3.0.patch
cd ${ROOT_DIR}/android-root/vendor/nxp/simw-top
mm -j$(nproc)
cd ${ROOT_DIR}/android-root/hardware/interfaces/keymaster/3.0/default
mm -j$(nproc)
```

3. AKM supports Various Auth Mechanism ,below are the list of supported Auth types:

```
None
PlatfSCP03
UserID
AESKey
ECKey
UserID_PlatfSCP03
AESKey_PlatfSCP03
ECKey_PlatfSCP03
```

4. By default SE05X Authentication is through None.For any other Auth type follow below steps:

```
cd ${ROOT_DIR}/android-root/vendor/nxp/simw-top
mm SE05X_Auth=(Auth Type) -j$(nproc)
eg. mm SE05X_Auth=PlatfSCP03 -j$(nproc)
cd ${ROOT_DIR}/android-root/hardware/interfaces/keymaster/3.0/default
mm -j$(nproc)
```

5. After successful build copy `keymaster_flash.bat` located at `$ROOT_DIR/android-root/vendor/nxp/simw-top/scripts/android/aosp_based` to `$ROOT_DIR/android-root/out/target/product/<BOARD_NAME>`. Execute the batch script to push all the necessary files onto the target board.
6. Other way to build SE050 based android keymaster is as follows:

```
cd ${ROOT_DIR}/android-root
export ANDROID_ROOT=$(pwd)
cd ${ANDROID_ROOT}/vendor/nxp/simw-top/scripts/android/aosp_based
source board_config.sh hikey960
./setup_script.sh
```

7. A batch script `keymaster_flash.bat` will be copied to `$ROOT_DIR/android-root/out/target/product/<BOARD_NAME>`. Execute the batch script to push all the necessary files onto the target board.

## Extract Secure Element Information

Refer to *SE Platform Information on Android platform*.

## Rotate Platform SCP03 Keys

Project `se05xRotatePlatfSCP03` is available to update Platform SCP03 keys on the SE. Build the project with build configuration `SE05X_Auth=PlatfSCP03`. For details about the tool, refer to *SE05X Rotate PlatformSCP Keys Demo*.

After building the project, push the built binary on the android device using `adb` tool and run it from the command line.

## How to use own Platform SCP03 Keys

Refer to [Section 9.11 Using own Platform SCP03 Keys](#) for details on how to use your own Platform SCP03 keys.

---

**Note:** Be sure to apply `keymaster_sepolicy.patch` to allow Platform SCP03 keys access to keymaster service.

---

## Retrieve Existing Certificates

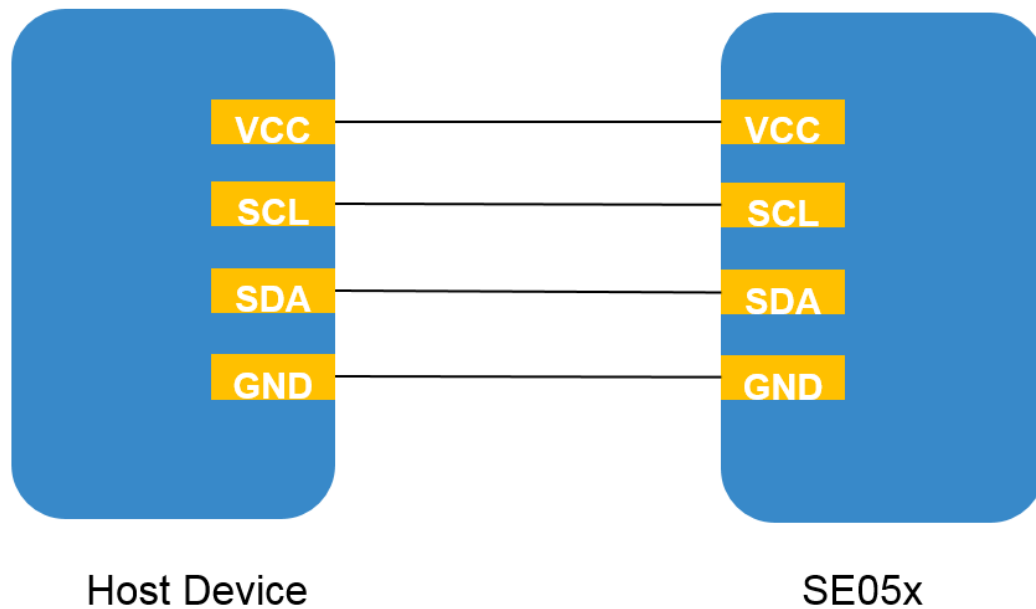
Refer to *Get Certificate from the SE*.

## How To Enable Logging

By default, information logs, error logs and warning logs are enabled but debug logs are disabled. To enable debug logs define `NX_LOG_ENABLE_DEFAULT_DEBUG` as 1 in `$ROOT_DIR/simw-top/hostlib/hostLib/libCommon/infra/nxLog_DefaultConfig.h`

## I2C connections with SE05x

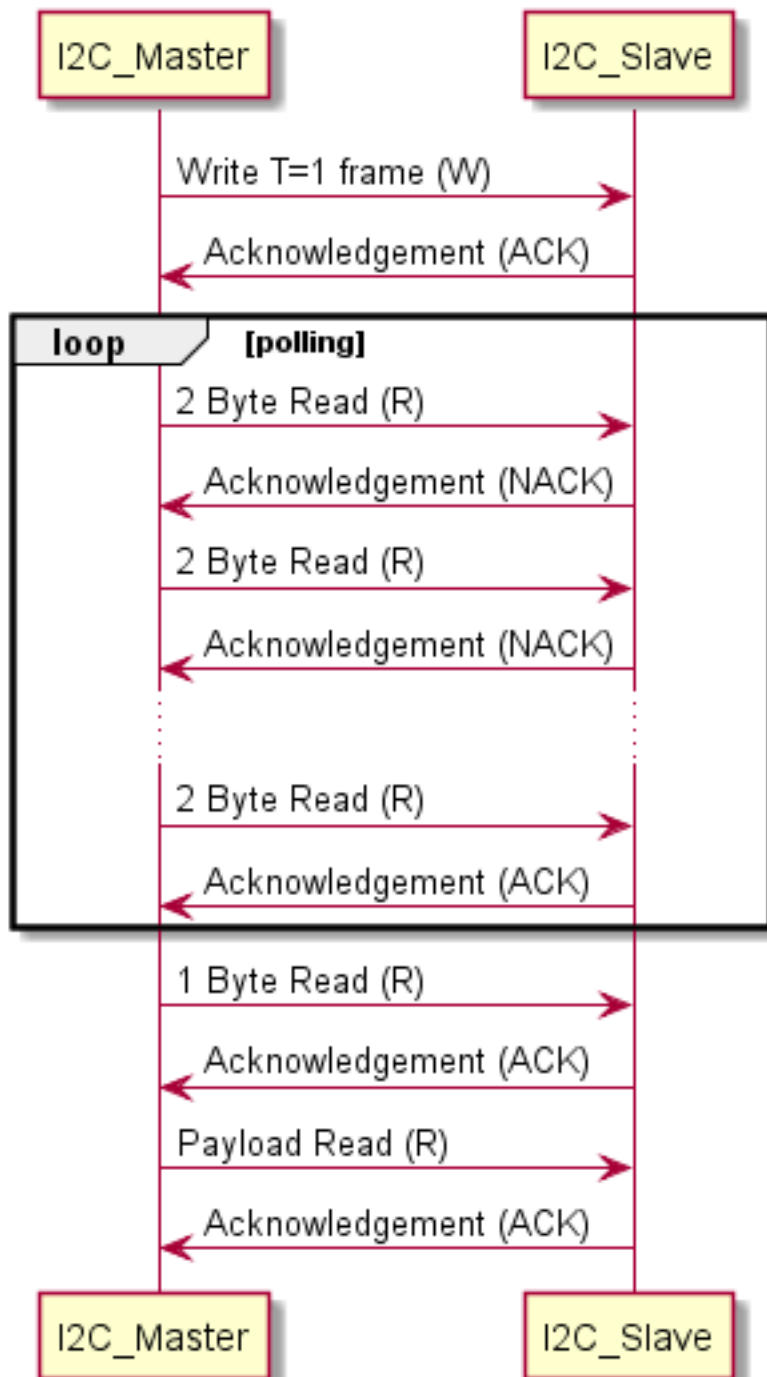
1. Below Diagram shows the wiring connection between Host Device and SE05x



---

#### I2C data transceive operation

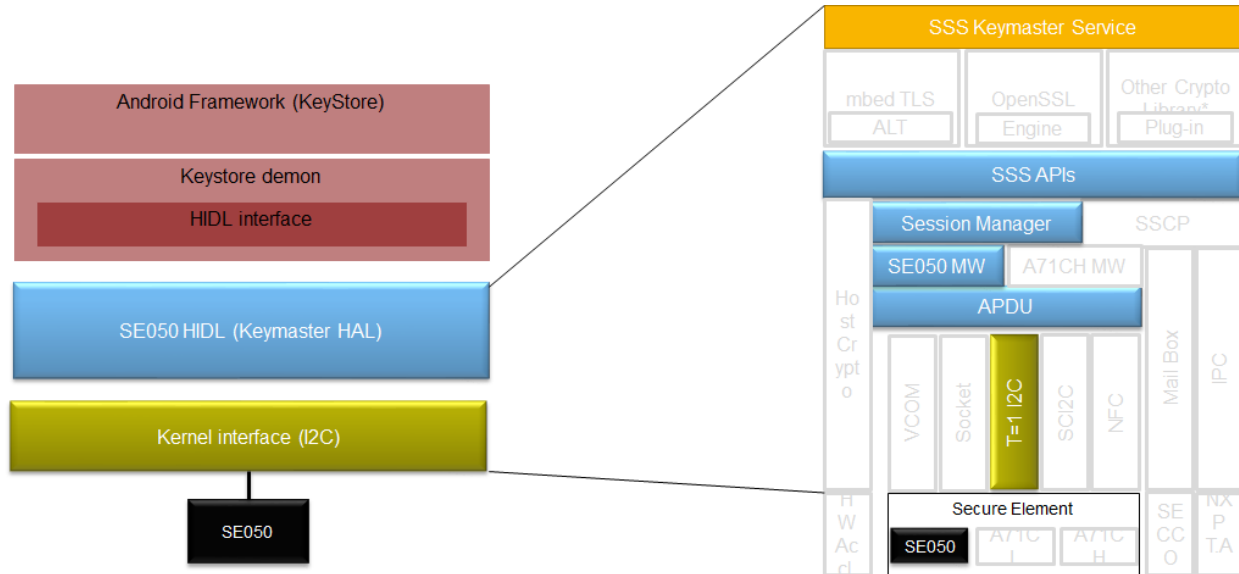
1. The Host Device acts as an I2C\_master while SE05x shall be the I2C\_slave.
2. HD transmits requested frame from applicaton layer to SE over I2C Bus. SE sends acknowledgement (ACK/NACK) for the received frame.
3. SE processes the recieved frame and prepares the response accordingly. HD polls for Read till the time Response is prepared and sent over I2C bus.
4. Following Diagram demonstrate the same.

**T1oI2C data Transreceive**

## Stack with SE050

We use Android Keymaster with SE050 over T=1 I2C. In this setup, we use physical T=1 over I2C Connection to the Applet.

The Architecture looks like this:



## Trust Provisioned keys

The trust provisioned SE contains ECC-256 and RSA-2048 keys. These keys are provisioned at specific keyIDs. In order to use these keys, we need to pass a magic number along with the corresponding keyID of the key to the keymaster import\_key API. Only when the import\_key parses the key and finds the magic as a part of the key, it returns the key blob of the trust provisioned key.

## Using TP RSA key

To use trust provisioned RSA key, pass the key in the following format:

```
modulus:
 a5:a6:b5:b6:a5:a6:b5:b6:xx:xx:xx:xx:xx:xx:xx:xx:
 xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:....
publicExponent: 65537 (0x10001)
privateExponent:
 A5:23:00:67:02:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:
 xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:....
prime1:
 xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:....
prime2:
 xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:....
exponent1:
 xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:....
exponent2:
 xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:....
coefficient:
 xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:....
```



Note here that the key modulus starts with the magic A5:A6:B5:B6:A5:A6:B5:B6 and the privateExponent starts with A5 followed by the 32-bit keyID (here, 0x23006702) of the trust provisioned RSA keypair. When an RSA key with modulus starting with the magic and privateExponent starting with A5 is passed to `import_key`, the RSA key stored at the corresponding keyID (0x23006702) is returned. An example of java code to import RSA keypair in this format is:

```
PrivateKey ImportTPKeyRSA() throws NoSuchAlgorithmException,
 InvalidKeySpecException
{
 PrivateKey privKey;
 KeyFactory KeyFac;
 BigInteger Mod, PrivExp, PubExp, PrimeP, PrimeQ, PrimeExpP, PrimeExpQ,
 ↪CrtCoef;
 RSAPrivateCrtKeySpec spec;
 try {
 KeyFac = KeyFactory.getInstance("RSA");
 } catch (NoSuchAlgorithmException e) {
 throw e;
 }
 Random rnd = new Random();
 StringBuffer temp = new StringBuffer(112);
 for(int i=0 ; i<112 ; i++)
 {
 int val = rnd.nextInt(16);
 temp.append(Integer.toString(val, 16));
 }
 StringBuffer buf = new StringBuffer(128);
 String mag = "a5a6b5b6a5a6b5b6";
 buf.append(mag);
 buf.append(temp);
 String mod = buf.toString();

 Mod = new BigInteger(mod, 16);
 PrivExp = new BigInteger("A523006702", 16); //KeyID in hex at which
 ↪Trust provisioned key is stored
 PubExp = new BigInteger("65537");
 PrimeP = new BigInteger("1");
 PrimeQ = new BigInteger("1");
 PrimeExpP = new BigInteger("1");
 PrimeExpQ = new BigInteger("1");
 CrtCoef = new BigInteger("1");

 // Create a RSA private key spec using components which have the magic and
 ↪keyID
 spec = new RSAPrivateCrtKeySpec(Mod, PubExp, PrivExp, PrimeP, PrimeQ,
 ↪PrimeExpP, PrimeExpQ, CrtCoef);

 try {
 // Generate a dummy keypair using key factory which will be in the
 ↪desired format to export trust provisioned keypair
 privKey = KeyFac.generatePrivate(spec);
 } catch (InvalidKeySpecException e) {
 throw e;
 }

 return privKey;
}
```

(continues on next page)

(continued from previous page)

```

void SetTPKeyRSA(String Label) throws NoSuchAlgorithmException,
↳InvalidKeySpecException, KeyStoreException,
 CertificateException
{
 PrivateKey privKey;
 X509Certificate cCert;
 // Dummy RSA certificate to create a keystore entry
 final String cDummyCert =
↳"-----BEGIN CERTIFICATE-----\nMIIB9zCCAWCgAwIBAgIEITKMnTANBgkqhkiG9w0BAQsFADA3MRgwFgYDVQDEw9JbnRl
↳

 try
 {
 privKey = ImportTPKeyRSA();
 }
 catch (Exception e)
 {
 throw e;
 }

 Certificate[] aUseCert;
 aUseCert = new X509Certificate[1];
 CertificateFactory cCertFac;
 InputStream in = new ByteArrayInputStream(cDummyCert.getBytes());
 try {
 cCertFac = CertificateFactory.getInstance("X.509");
 } catch (CertificateException e) {
 throw e;
 }
 aUseCert[0] = (X509Certificate) cCertFac.generateCertificate(in);

 try
 {
 // Store the keypair in keystore with alias=Label and dummy_
↳certificate chain = aUseCert
 m_cKeyStore.setKeyEntry(Label, (Key) privKey, null, aUseCert);
 }
 catch (KeyStoreException e)
 {
 throw e;
 }
 return;
}

```

## Using TP EC key

To use trust provisioned EC key, pass the key in the following format:

```

priv:
 a5:a6:b5:b6:a5:a6:b5:b6:c3:02:00:01:xx:xx:xx:
 xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:....
pub:
 xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:....

```

Note here that the private component of the EC keypair contains the magic A5:A6:B5:B6:A5:A6:B5:B6 followed

by the 32-bit keyID (here, 0xC3020001) of the trust provisioned EC keypair. When an EC key with private component starting with the magic is passed to `import_key`, the EC keypair stored at the corresponding keyID (0xC3020001) is returned. An example of java code to import EC keypair in this format is:

```
ECPrivateKey ImportTPKeyECC() throws NoSuchAlgorithmException,
 InvalidKeySpecException,
 InvalidParameterSpecException,
 InvalidAlgorithmParameterException, NoSuchProviderException
{
 ECPrivateKey privKey;
 KeyFactory KeyFac;
 BigInteger PrivS;
 ECPParameterSpec ECSpec;
 ECPrivateKeySpec PrivSpec;
 Random rnd = new Random();

 StringBuffer temp = new StringBuffer(40);
 for(int i=0 ; i<40 ; i++)
 {
 int val = rnd.nextInt(16);
 temp.append(Integer.toString(val, 16));
 }

 StringBuffer buf = new StringBuffer(64);
 String mag = "a5a6b5b6a5a6b5b6";
 String keyobject = "c3020001";
 buf.append(mag);
 buf.append(keyobject);
 buf.append(temp);
 String magic = buf.toString();
 PrivS = new BigInteger(magic, 16);
 String cECCCurveName = "secp256r1";
 AlgorithmParameters algSpec = AlgorithmParameters.getInstance("EC");

 try {
 algSpec.init(new ECGenParameterSpec(cECCCurveName));
 } catch (InvalidParameterSpecException e1) {
 e1.printStackTrace();
 }

 ECSpec = algSpec.getParameterSpec(ECPParameterSpec.class);
 // Create PrivateKey spec with parameters for curve secp256r1 and private key_
 ↪containing the magic and the keyID
 PrivSpec = new ECPrivateKeySpec(PrivS, ECSpec);

 try {
 KeyFac = KeyFactory.getInstance("EC");
 } catch (NoSuchAlgorithmException e) {
 throw e;
 }

 try {
 // Generate a dummy keypair using key factory which will be in the_
 ↪desired format to export trust provisioned keypair
 privKey = (ECPrivateKey) KeyFac.generatePrivate(PrivSpec);
 } catch (InvalidKeySpecException e) {
 throw e;
 }
}
```

(continues on next page)

(continued from previous page)

```

 return privKey;
 }

 void SetTPKeyEC(String Label) throws NoSuchAlgorithmException,
 ↳InvalidKeySpecException, KeyStoreException,
 CertificateException, GeneralSecurityException {
 ECPrivateKey privKey;
 X509Certificate cCert;
 final String cDummyCert = "-----BEGIN CERTIFICATE-----\n" +
 "MIIBeTCCASCgAwIBAgIJAKtU6mCCLJeyMAoGCCqGSM49BAMCMBExDzANBgNVBAMM
 ↳BmRlbW9DQTA

 privKey = ImportTPKeyECC();

 Certificate[] aUseCert;
 aUseCert = new X509Certificate[1];
 CertificateFactory cCertFac;
 InputStream in = new ByteArrayInputStream(cDummyCert.getBytes());

 try {
 cCertFac = CertificateFactory.getInstance("X.509");
 } catch (CertificateException e) {
 throw e;
 }
 aUseCert[0] = (X509Certificate) cCertFac.generateCertificate(in);

 try
 {
 // Store the keypair in keystore with alias=Label and dummy
 ↳certificate chain = aUseCert
 m_cKeyStore.setKeyEntry(Label, (Key) privKey, null, aUseCert);
 }
 catch (KeyStoreException e)
 {
 throw e;
 }

 return;
 }
}

```

## 9.6 How to get SE Platform Information and UID

Follow any one of the following methods to get the SE platform information.

### 9.6.1 Using TeraTerm and pre-built binary

A pre-built binary for `se05x_Get_Info` is available in `binaries` directory for FRDM-K64F and iMX-RT1050.

You would need to install a serial terminal application, like TeraTerm.

- Install TeraTerm on your system. To setup TeraTerm, refer
- Flash the pre-built binary on your hardware.
- View the logs on TeraTerm.

You would be able to see a log like this:

```
App:INFO :PlugAndTrust_v02.10.90_20190726
sss:INFO :atr <Len=35>
00 00 00 00 03 96 04 03 E8 00 FE 02 0B 03 E8 08
01 00 00 00 00 64 00 00 0A 4A 43 4F 50 34 20 41
54 50 4F
sss:WARN :Communication channel is Plain.
sss:WARN :!!!Not recommended for production use.!!!
App:WARN :*****
App:INFO :uid <Len=18>
04 0D 9 67 C 41 1B 5 B7 A
D
App:WARN :*****
App:INFO :Applet Major = 3
App:INFO :Applet Minor = 1
App:INFO :Applet patch = 0
App:INFO :AppletConfig = 67D2
App:INFO :Without ECDAA
App:INFO :With ECDSA_ECDH_ECDHE
App:INFO :Without EDDSA
App:INFO :Without DH_MONT
App:INFO :With HMAC
App:INFO :Without RSA_PLAIN
App:INFO :With RSA_CRT
App:INFO :With AES
App:INFO :With DES
App:INFO :With PBKDF
App:INFO :With TLS
App:INFO :Without MIFARE
App:INFO :With I2CM
App:INFO :SecureBox = 010B
App:WARN :*****
App:INFO :Tag value - proprietary data 0xFE = 0xFE
App:INFO :Length of following data 0x45 = 0x45
App:INFO :Tag card identification data <Len=2>
DF 28
App:INFO :Length of card identification data 0x46 = 0x42
App:INFO :Tag configuration ID 0x01 = 0x01
App:INFO :Length configuration ID 0x0C = 0x0C
App:INFO :Configuration ID <Len=12>
DE 02 76 92 E8 1D E6 08 F3 88 0D 5F
App:INFO :OEF ID <Len=2>
7 2
App:INFO :Tag patch ID 0x02 = 0x02
App:INFO :Length patch ID 0x08 = 0x08
App:INFO :Patch ID <Len=8>
00 00 00 00 00 00 00 01
App:INFO :Tag platform build ID1 0x03 = 0x03
App:INFO :Length platform build ID 0x18 = 0x18
App:INFO :Platform build ID <Len=24>
4A 33 52 33 35 31 30 32 34 44 30 30 31 31 30 30
F2 00 00 40 2D 7A 09 42
App:INFO :JCOP Platform ID = J3R3 24D0 00
App:INFO :Tag FIPS mode 0x05 = 0x05
App:INFO :Length FIPS mode 0x01 = 0x01
App:INFO :FIPS mode var = 0x01
App:INFO :Tag pre-perso state 0x07 = 0x07
App:INFO :Length pre-perso state 0x01 = 0x01
App:INFO :Bit mask of pre-perso state var = 0x01
App:INFO :Tag ROM ID 0x08 = 0x08
App:INFO :Length ROM ID 0x08 = 0x08
App:INFO :ROM ID <Len=8>
2E 5A D8 84 09 C9 BA DB
App:INFO :Status Word <SW> <Len=2>
90 00
App:INFO :ex_sss Finished
```

The highlighted log is the SE UID, OEF ID and JCOP Platform ID.

## 9.6.2 Using VCOM and binary

- Flash VCOM binary present in `binaries` directory according to your board.
- Check VCOM Port number in device manager.
- **Build project `se05x_Get_Info` with the following configuration:**
  - SMCOM: VCOM
  - Host: PCWindows
- Run the built binary as:

```
se05x_Get_Info.exe COMxx
```

Where *COMxx* is the VCOM port number obtained from step 2.

You would be able to see a log like this:

```

App:INFO :PlugAndTrust_v02.10.90_20190726
App:INFO :Running se05x_Get_Info.exe
App:INFO :Using PortName='COM16' (CLI)
Opening COM Port '\\.\COM16'
sss:INFO :atr (Len=35)
00 A0 00 00 03 96 04 03 E8 00 FE 02 0B 03 E8 08
01 00 00 00 00 64 00 00 0A 4A 43 4F 50 34 20 41
54 50 4F
sss:WARN :Communication channel is Plain.
sss:WARN :!!!Not recommended for production use.!!!
App:WARN :#####
App:INFO :uid (Len=18)
04 0D 9 67 C 41 1B 5 B7 A
D
App:WARN :#####
App:INFO :Applet Major = 3
App:INFO :Applet Minor = 1
App:INFO :Applet patch = 0
App:INFO :AppletConfig = 67D2
App:INFO :WithOut ECDAA
App:INFO :With ECDSA_ECDH_ECDHE
App:INFO :WithOut EDDSA
App:INFO :WithOut DH_MONT
App:INFO :With HMAC
App:INFO :WithOut RSA_PLAIN
App:INFO :With RSA_CRT
App:INFO :With AES
App:INFO :With DES
App:INFO :With PBKDF
App:INFO :With TLS
App:INFO :WithOut MIFARE
App:INFO :With I2CM
App:INFO :SecureBox = 010B
App:WARN :#####
App:INFO :Tag value - proprietary data 0xFE = 0xFE
App:INFO :Length of following data 0x45 = 0x45
App:INFO :Tag card identification data (Len=2)
DF 28
App:INFO :Length of card identification data 0x46 = 0x42
App:INFO :Tag configuration ID 0x01 = 0x01
App:INFO :Length configuration ID 0x0C = 0x0C
App:INFO :Configuration ID (Len=12)
DE 02 76 92 E8 1D E6 08 F3 88 0D 5F
App:INFO :OEF ID (Len=2)
7 2
App:INFO :Tag patch ID 0x02 = 0x02
App:INFO :Length patch ID 0x08 = 0x08
App:INFO :Patch ID (Len=8)
00 00 00 00 00 00 00 01
App:INFO :Tag platform build ID1 0x03 = 0x03
App:INFO :Length platform build ID 0x18 = 0x18
App:INFO :Platform build ID (Len=24)
4A 33 52 33 35 31 30 32 34 44 30 30 31 31 30 30
F7 00 00 40 2D 7A 09 42
App:INFO :JCOP Platform ID = J3R3 2400 00
App:INFO :Tag FIPS mode 0x05 = 0x05
App:INFO :Length FIPS mode 0x01 = 0x01
App:INFO :FIPS mode var = 0x01
App:INFO :Tag pre-perso state 0x07 = 0x07
App:INFO :Length pre-perso state 0x01 = 0x01
App:INFO :Bit mask of pre-perso state var = 0x01
App:INFO :Tag ROM ID 0x08 = 0x08
App:INFO :Length ROM ID 0x08 = 0x08
App:INFO :ROM ID (Len=8)
2E 5A D8 84 09 C9 BA DB
App:INFO :Status Word (SW) (Len=2)
90 00
App:INFO :ex_sss Finished

```

The highlighted log is the SE UID, OEF ID and JCOP Platform ID.

### 9.6.3 Using pySSSCLI Tool

- Flash VCOM binary present in `binaries` directory according to your board.
- Check VCOM Port number in device manager.
- Run the following commands in `binaries/pySSSCLI` directory:

```
ssscli.exe connect se050 vcom COMxx
ssscli.exe se05x uid
ssscli.exe disconnect
```

Where *COMxx* is the VCOM Port number obtained in step 2.

You would be able to see a log like this:

```
Opening COM Port '\\.\COM5'
sss:INFO :atr (Len=35)
 00 A0 00 00 03 96 04 03 E8 00 FE 02 0B 03 E8 08
 01 00 00 00 00 64 00 00 0A 4A 43 4F 50 34 20 41
 54 50 4F
sss:WARN :Communication channel is Plain.
sss:WARN :!!!Not recommended for production use.!!!
INFO:sss.se05x:04005001222cb5ffe83a52042f0559550000
INFO:sss.se05x:Unique ID: 04005001222cb5ffe83a52042f0559550000
```

The highlighted log is the SE UID

### 9.6.4 SE Platform Information on Android platform

1. SE will have preconfigured information with which one can verify the functionalities it supports.
2. Follow the steps in *AOSP build Environment Setup* as a pre-requisite.
3. After successful compilation of Android keymaster, `se05xGetInfo.bin` will get generated in `$ROOT_DIR/android-root/out/target/product/<BOARD_NAME>/testcases/se05xGetInfo/arm` and `$ROOT_DIR/android-root/out/target/product/<BOARD_NAME>/testcases/se05xGetInfo/arm64` directories.
4. To get the platform information on Android, follow below steps:

```
adb root && adb wait-for-device && adb remount
adb push testcases/se05xGetInfo/arm/se05xGetInfo /system/vendor/bin/se05xGetInfo
adb reboot
adb root && adb wait-for-device && adb remount
adb shell
cd system/vendor/bin
./se05xGetInfo
```

5. In `adb logcat | grep "NXPKeymasterDevice"` respected information will be shown.

An example log of SE platform information is given below.



```

NXPKeymasterDevice: App: PlugAndTrust_v02.10.02_20190809
NXPKeymasterDevice: App: Running ./se05x_Get_Info
NXPKeymasterDevice: App: If you want to over-ride the selection, use ENV=EX_SSS_BOOT_SSS_PORT or pass in command line arguments.
NXPKeymasterDevice: sss:atr (Len=35) 00A0000003960403E800FE020B03E80801000000006400000A4A434F5034204154504F
NXPKeymasterDevice: sss: Communication channel is Plain.
NXPKeymasterDevice: sss: !!!Not recommended for production use!!!
NXPKeymasterDevice: App: #####
NXPKeymasterDevice: App:uid (Len=18) 040D 967C 411B 5B7A D
NXPKeymasterDevice: App: #####
NXPKeymasterDevice: App: Applet Major = 3
NXPKeymasterDevice: App: Applet Minor = 1
NXPKeymasterDevice: App: Applet patch = 0
NXPKeymasterDevice: App: AppletConfig = 67D2
NXPKeymasterDevice: App: Without ECDAA
NXPKeymasterDevice: App: With ECDSA_ECDH_ECDHE
NXPKeymasterDevice: App: Without EDDSA
NXPKeymasterDevice: App: Without DH_MONT
NXPKeymasterDevice: App: With HMAC
NXPKeymasterDevice: App: Without RSA_PLAIN
NXPKeymasterDevice: App: With RSA_CRT
NXPKeymasterDevice: App: With AES
NXPKeymasterDevice: App: With DES
NXPKeymasterDevice: App: With PBKDF
NXPKeymasterDevice: App: With TLS
NXPKeymasterDevice: App: Without MIFARE
NXPKeymasterDevice: App: With I2CH
NXPKeymasterDevice: App: SecureBox = 010B
NXPKeymasterDevice: App: #####
NXPKeymasterDevice: App: Tag value - proprietary data 0xFE = 0xFE
NXPKeymasterDevice: App: Length of following data 0x45 = 0x45
NXPKeymasterDevice: App:Tag card identification data (Len=2) DF28
NXPKeymasterDevice: App: Length of card identification data 0x46 = 0x42
NXPKeymasterDevice: App: Tag configuration ID 0x01 = 0x01
NXPKeymasterDevice: App: Length configuration ID 0x0C = 0x0C
NXPKeymasterDevice: App:Configuration ID (Len=12) DE027692E81DE608F3880D5F
NXPKeymasterDevice: App:OEF ID (Len=2) 7 2
NXPKeymasterDevice: App: Tag patch ID 0x02 = 0x02
NXPKeymasterDevice: App: Length patch ID 0x08 = 0x08
NXPKeymasterDevice: App:Patch ID (Len=8) 0000000000000001
NXPKeymasterDevice: App: Tag platform build ID1 0x03 = 0x03
NXPKeymasterDevice: App: Length platform build ID 0x18 = 0x18
NXPKeymasterDevice: App:Platform build ID (Len=24) 4A335233353130323444303031313030F70000402D7A0942
NXPKeymasterDevice: App: JCOP Platform ID = J3R3 24D0 00
NXPKeymasterDevice: App: Tag FIPS mode 0x05 = 0x05
NXPKeymasterDevice: App: Length FIPS mode 0x01 = 0x01
NXPKeymasterDevice: App: FIPS mode var = 0x01
NXPKeymasterDevice: App: Tag pre-perso state 0x07 = 0x07
NXPKeymasterDevice: App: Length pre-perso state 0x01 = 0x01
NXPKeymasterDevice: App: Bit mask of pre-perso state var = 0x01
NXPKeymasterDevice: App: Tag ROM ID 0x08 = 0x08
NXPKeymasterDevice: App: Length ROM ID 0x08 = 0x08
NXPKeymasterDevice: App:ROM ID (Len=8) 2E5AD88409C9BADB
NXPKeymasterDevice: App:Status Word (SW) (Len=2) 9000
NXPKeymasterDevice: App: ex_sss Finished

```

The highlighted log is the SE UID, OEF ID and JCOP Platform ID.

## 9.7 Version Information

Item	Version Number
Release Version	v02.14.00_20200403
Middleware	v02.14.00_20200403
SSS APIs	v03.01.00_20200403
Demos and Use Cases	v02.14.00_20200403
JCOP	Platform ID J3R351021E950400
SE050 Applet	03.01
A71CH Applet	1.3

## 9.8 Certificate Chains : ROOT

- *ECC*
  - *ROOT CA*
  - *Intermediate CA*
- *RSA*
  - *ROOT CA*
  - *Intermediate CA*

The directory `demos/Certificate_Chains/ROOT` contains RootCA and Intermediate Certificates used in various configurations of SE050.

### 9.8.1 ECC

This directory contains the ECC chain of trust for cloud on-boarding.

#### ROOT CA

The file `IOT_NXP-01-CERT_IOT_CA_KEY-IoTRootCAvE305-01-20190320162439-EC_SEC_P384R1-4B7E5A.crt` contains ROOT CA.

```
-----BEGIN CERTIFICATE-----
MIIB1jCCAVmgAwIBAgIBATAMBggqhkjOPQQDAwUAMEEeXfzAVBgNVBAsMDlBsdWcg
YW5kIFRydXN0MQwwCgYDVQQKDANOWFAxGDAWBgNVBAMMD05YUCBSb290Q0F2RTMw
NTAeFw0xOTAzMjAxNTI2MDRaFw0zNzAzMjAxNTI2MDRaMEEeXfzAVBgNVBAsMDlBs
dWcgYW5kIFRydXN0MQwwCgYDVQQKDANOWFAxGDAWBgNVBAMMD05YUCBSb290Q0F2
RTMwNTB2MBAGByqGSM49AgEGBSuBBAAiA2IABNRrkWrw7iwM3oTw1Ay8I1yzOF+g
OTNfZqfn/93N1xcK8kNEA8wNuuVjzsDXKHx7O1jrGZfy7YLjKYTwZR5wURXrE7lp
PIwwdWE3OokTmhiMiFXnzSgSHCgtb+VF6nv76aMjMCEwDwYDVR0TAQH/BAUwAwEB
/zAOBgNVHQ8BAf8EBAMCAQYwDAYIKoZIzj0EAwMFANpADBMAjEA7cCFz6PcBHKi
HRtbE3qi0Lj9iqxe8/2w8XrLclAYhpMzZQOC05j3ZmgJ22B1bLk3AjEAhVnhuawO
Zh1Jqwf7zySh/rNJezFhGTyHAjQ9tTfY9eZAdc25feEN4j/Ad+7TF1Rg
-----END CERTIFICATE-----
```

#### Intermediate CA

The file `IOT_NXP-01-CERT_IOT_4LAYER_CA_KEY-IoTInt4LAYERCAvE205-01-20190320164314-EC_SEC_P256R1.crt` contains the Intermediate CA.

```
-----BEGIN CERTIFICATE-----
MIIBYjCCAU6gAwIBAgIBBDAMBggqhkjOPQQDAgUAMEEeXfzAVBgNVBAsMDlBsdWcg
YW5kIFRydXN0MQwwCgYDVQQKDANOWFAxGDAWBgNVBAMMD05YUCBSb290Q0F2RTMw
NTAeFw0xOTAzMjAxNTQyNTRaFw0zNDAzMjAxNTQyNTRaMFAXfzAVBgNVBAsMDlBs
dWcgYW5kIFRydXN0MQwwCgYDVQQKDANOWFAxJzAlBgNVBAMMHk5YUCBJbnRlcm1l
ZGlhdGUtNEhewVvyQ0F2RTIwNTBZMBMGBYqGSM49AgEGCCqGSM49AwEHA0IABO+d
HK3Oa4FnkKN9/1JQpR2KarpXmwNRaEG6Zb+kzTwnXRw4Cvknd8IAcjXHqb93VfX5
rO+4MjX/gzqYagJByWejJjAkMBIGA1UdEwEB/wQIMAYBAf8CAQEwDgYDVR0PAQH/
BAQDAgEGMAwGCCqGSM49BAMCBQADaAAwZQIxAPvXuvlW+zksBbz0NyyzpgFP1rCj
-----END CERTIFICATE-----
```

(continues on next page)

(continued from previous page)

```

IZOHpfZhaERw/DEj+4aAESa5vgtiR3CspIP5pAIwRsD1Z9fdBnPSlaVMmNAXjAlZ
FPhbV7A0OXydSYhI8M5uTENrdqzvsYYr2jfqIEcp
-----END CERTIFICATE-----

```

## 9.8.2 RSA

This directory contains the RSA chain of trust for cloud on-boarding.

### ROOT CA

The file `IOT_NXP-01-CERT_IOT_CA_KEY-IoTRootCAvR406-01-20190425163255-RSA4096-BAB872.crt` contains the ROOT CA.

```

-----BEGIN CERTIFICATE-----
MIIFIDCCAwiGAwIBAgIBATANBgkqhkiG9w0BAQwFADBBMRcwFQYDVQQLEDA5QbHVn
IGFuZCBUCnVzdDEMMAoGA1UECgwDTlhQMrgwFgYDVQQDDA9OWFAGUm9vdENBdlI0
MDYwHhcNMjkwNDI1MTQzMzAzWWhcNMzcwNDI1MTQzMzAzWjBBMRcwFQYDVQQLEDA5Q
bHVnIGFuZCBUCnVzdDEMMAoGA1UECgwDTlhQMrgwFgYDVQQDDA9OWFAGUm9vdENB
dlIOMDYwggiMA0GCSqGSIb3DQEBAQUAA4ICDwAwggIKAoICAQCwRBRveWxzoVln
bFOWxhjFmX6hgPBB5o7pOVVGqvvdKsvcdZh7+hozTPzI7d5eCJWtiIYZ4xUXxiP8
MttIsssT4yrZjpSy2qTWbn5T/1B7a2e/A5JTGUCX05/uFATfZTs2pcoUydb68PnB
LVsS8ilatpxN4Tiy5NPchWM2mL7YwoKJxCNgZF5WYxvm4OSohMy7rAF010ujy1k
6L/XydcSv48G6ZlpCYH8Tn5cV7UEuNWpggq+0Wgpz6xz/ZHyfTxKhuBtj4eh3GyDv
Fmt1L1hJT/k17+Q19rW3U7OBV5w8ehuLoRNsNzG2zc9rFh1K9pVX9DRZQKI5XNoh
1he7oSxJfKjsLn27w0fOHraOaefrYuOrbKDA/X+cJzyKOVxWNhVyFRryn2i058Wa
HuBqja7fnubexMP1Mr2b/hZV1lSPmFVDk+B17VXpvc/tYXeq1SK1Tj/1gZ5tpmpm
SBBqcHsp0YQEKNLAnG+K9sFOkamf1Yt0IO/iyoYPUQ5IV644pOiztO2myFrT04+K
Pwi+XwoRBVSK/WrI5vy16HzAJqPP1Nb6QKdOelQ+NwNMyUySFzHavd3L4XaiNP4Z
iPgjSG/me8ozcPTFR29eibPl7p9YlhocmU4g2UURStVL0cpdOrc36BEv6DQ6RN4H
7N9VM9AIP+1047argR+ciYzkY+Xd+wIDAQABoyMwITAPBgNVHRMBAf8EBTADAQH/
MA4GA1UdDwEB/wQEAWIBBjANBgkqhkiG9w0BAQwFAAOCAgEAQsNB0/011ELP5rHN
rk9+sTh+Mr6Ye7geTYT1QRCCpLuyUuyC3O51dSOKP3RorsNYD/aCJGHdvh0ofyj
yKx5lre2+b2foSVg+ZjQQf3qju+7bqh3tWld2bI49+yRoCIplJZQ8V91ReH+33k
TzYNv/tvzYxnF8AF7wJ93zyafXJvIQ6AUnVoDdgtN203M853jxavjZHSVjTQXaR0
LwYCCNiE+bE2Z4owjerG3QAVG/ju8Xz+bhOFry/+M1ib1F8KGZRzSA0E0ijMhhfZ
ZIP6eXjWWaoL+9iqD8F8/IXxW2zH37IzOTB6s2xcSFvjGbzVJ7CHy8/c4OQEoZpi
gujcPF5axOpI2RSnUPX+YFyu3od5URS3ybo9aT7b1CLURkjbhcGHCo04NL3h6Zg1
G2ktasOCL3moypWY0EYNwrHKyccVY3VVR1H89P03qQf5uUAADFcrpZ81B7yPy1mt
4qcZfbACzUt300UzUI1VDyHL6NcTw/1Da26aGKZMseHan6xESDpiHMC5efv4Xd7U
k+bVedNwTKooDzn1K5WKTiZ3nUcSRAhdmiaQrAb6lKYEcDxRK5VPtr/MU6cL2PKZ
m9oZ0hnfb5EbYJH4tWaite7j5KMrto1tT+woeWP/dSmDg31ZCBmc3Sx0NaU12jQZ
2vnNPomcaE8BkvyrhEPkSw/VXTw=
-----END CERTIFICATE-----

```



### 9.9.1 Certificate Chain for SE050

The directory `demos/Certificate_Chains/0004_A1F4` contains Intermediate Certificates used in Dev Kit configurations. It is signed by *Certificate Chains : ROOT*

#### Intermediate CA : ECC

The file `CloudConn-Intermediate-ECC_OEF_A1F4.crt` is certificate used for Dev Kit configuration.

```
-----BEGIN CERTIFICATE-----
MIIBujCCAWGgAwIBAgIKBABQAQAEofQAADAKBggqhkjOPQQDAjBQMRcwFQYDVQQQL
DA5QbHVnIGFuZCBUCnVzdDEMMaGA1UECgwDTlhQMScwJQYDVQQDDDB5OWFAgSW50
ZXJtZWZrYXR1LlRMYXl1ckNBdkUyMDUwHhcNMtkWnJE4MDAwMDAwWhcNMzQWnJE0
MDAwMDAwWjBdMRcwFQYDVQQQLDA5QbHVnIGFuZCBUCnVzdDEMMaGA1UECgwDTlhQ
MTQwMgYDVQQDDCtDbG91ZENvbW4tSW50ZXJtZWZrYXR1LTA0MDA1MDAxMDAwNEEEx
RjQtRUNDMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAENi7KN2Z6tbFuhjRrqqHb
xzUWSCUrdPmFP8xpsIOEHM8WJPvj8D+MCeu6Y2WU/ILhxp1Y3TJ2hGZtV4foO3pq
NqMWMBQwEgYDVR0TAQH/BAgwBgEB/wIBADAKBggqhkjOPQQDAgNHADBEAiB0waP1
4dcbmqbLUfQRscOPVzSeqT2dfdwg0W0g/FlaPgIgS2ppnaRASzc5rCoeVk5AeMb2
j7IKeHDyeMRP0wlapU=
-----END CERTIFICATE-----
```

#### Intermediate CA : RSA

The file `CloudConn-Intermediate-RSA_OEF_A1F4.crt` is certificate used for Dev Kit configuration.

```
-----BEGIN CERTIFICATE-----
MIIFRzCCAy+gAwIBAgIKBABQAQAEofQAADANBgkqhkiG9w0BAQsFADBQMRcwFQYD
VQQQLDA5QbHVnIGFuZCBUCnVzdDEMMaGA1UECgwDTlhQMScwJQYDVQQDDDB5OWFAg
SW50ZXJtZWZrYXR1LlRMYXl1ckNBdkUyMDUwHhcNMtkWnJE4MDAwMDAwWhcNMzQW
NjE0MDAwMDAwWjBdMRcwFQYDVQQQLDA5QbHVnIGFuZCBUCnVzdDEMMaGA1UECgwD
TlhQMTQwMgYDVQQDDCtDbG91ZENvbW4tSW50ZXJtZWZrYXR1LTA0MDA1MDAxMDAw
NEEExRjQtU1NBMIICiANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAAzKNa+y5n
9CQK9Aw4F1Lrml7chE3CBFgnp49OSNII0mI2G/YmhI2zotydyqIpl7Wk+gJd1wpQM
rKuh2RmLihoIyLmuuOVxKbn1Tw35tshiIj48IinJ7G6n4x7SmU2kHjPqsjdF4ncp
VK6IcXoNQ+zten8/q5Pcbmozo+UriUhLGdu6By/Z3+BseTsNYJhouIzN8CwZ1h+r
LRgjpiodrwHE0jVbe74Fqjcb7TEqACHZNEaChlfnSs4BaiiMOSMGgGhGrPMHxb0z
VHESSJGZ3aa7xzFakoP6hNz6m82q14WYMMM/2fbuOh2mIgrEexz22zTGsG1a3+wt
uy46QVCivEMwsMDYzOjoc0lCbAsRUft9pbzdKVXt5qApPQLxOzSkGduOw5N/jdN
Kmt/sa7gh8uRqnQVvzjKsmDg0TqolPuTrt2tS37lMwlyKToAeo09w/HkMb6CF7GB
EgTTnob1MGUpJRdk9MM/PMi15tsXT30G1MVcXoRdk6y9nBsXLAPzBC3SQAjmk6T1
KWPgBlrjNIFmkosCKQPPznldygpbcCKHeRt29X+TsQq12tDTHMUTdG5NWJ/6gPEk
2url2J2iEF9LaAu3dM5aKfLauoyvKen1Z6FUBVQz9PgLRrcCR52eiUn6AOR9gZZN
Lozp/uQQU+CBMw0daEBI7+xethA9F426CCMAwEAAaMWMBQwEgYDVR0TAQH/BAgw
BgEB/wIBADANBgkqhkiG9w0BAQsFAAOCAgEAAJLw+RCn/qhLVPni7NwkTfi69bnrF
jE6nhhgveleZ71xUPu3Jdd+swgfdZmKdNUfrr4mQJ21/rj2Z4yx7WA7Cy3IjG52d
9A9/jrQU8T9AmqME9e4rwTmJzCtKx91yFex+jcisJphcc6zkoDmp/XaumXi9GW
M+J0laY+hj6nw9BWbd/wGDxe2mzBjdWNTp8jb3CKuqJQexWB5pjNUnexdJSDFVXs
Hf0NYtolzK7dh+/GMUfdTCX/8g3hN2Won1DD6/ftXdgCSXExaBz0EhsethdFb3y6
vR+MvngMoZhbyGH5VzQ2sESo9vGzfPSiM8mHvjaV++hDduVEpdoxSE/C8TdNpUZ7
Lh6ElJYcUbDE2VV93/dYaiM5Cag/SuKG/8phZ3g9eXT9oPmOEtKXg9bdMJoNwJ0N
LiAW3eFdBBmDGB/6D5vSOD3YkPhtBUIC+/9vBL0KNx9V9/4xWZAnmTtjLfyLoyst
IE5Qnr0TpqZ123+Tvj9X7ulNUuu7MMHtiAs2SKM9iWMXAd/UW6Vq5x7SKjQ7b6FQ
vCn/CAZn9uV4ixwDzbIoQV+4ps9QuyqVGJbtqEctxYo896ulTeJAFyiPnfeF1WdM
gPmab4mVjTyikESCTskM2nNYVkkQQ5ANHfEQ8XVoRsdJmp3rMl6MD1n1zjfSyEHRQ
```

(continues on next page)

(continued from previous page)

```
1XL23wm/K2UEOfw=
-----END CERTIFICATE-----
```

**Note:** The certificates shown here were last Updated on June 18, 2019 in this page/document. These certificate chains are only for Dev Kit configuration (OEF A1F4).

## 9.10 JRCP\_v1 Server

This section explains how the JRCP\_v1 server enables a client process (remote or local) to establish a communication session with a Secure Element.

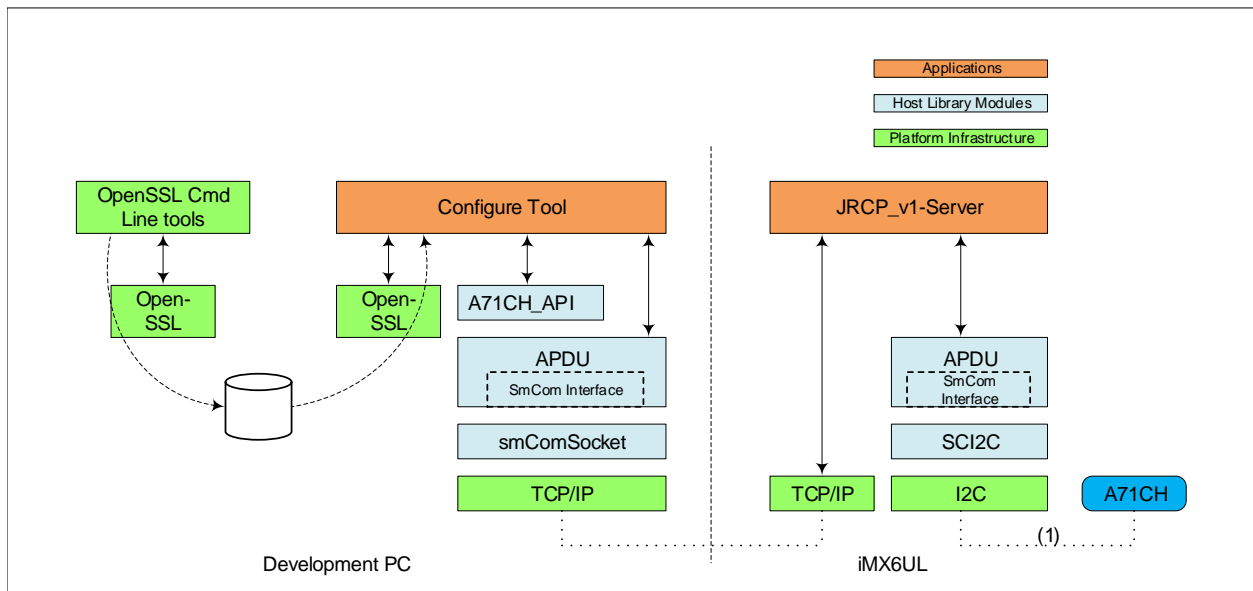
**Note:** The JRCP\_v1 server was previously known as RJCT server.

### 9.10.1 Introduction

The JRCP\_v1 server is a standalone process that can establish a communication session with a Secure Element on behalf of a client process. Only one client process (at a time) can have a communication session via the JRCP\_v1 server with the Secure Element.

The JRCP\_v1 server listens for incoming connection requests from client processes on TCP/IP port 8050. The client application must be compiled for the JRCP\_V1 smCom protocol layer.

The following figure illustrates a client application (in the example the Configure Tool) connecting to the JRCP\_v1 server,



The tool is provided in source code (.../hostlib/rjct). It can be built to connect to the Secure Element with e.g. the SCI2C or the T=IoI2C protocol.

## 9.10.2 Using the JRCP\_v1 server

The JRCP\_v1 server application that is bundled with the Host SW is a command line application. In the following example the server is started from a shell on the embedded platform

```
root@imx6ulevk:~# ./jrctp1_server
RemoteJCShell Terminal Server (supporting SCI2C) Rev 0.92
A71 (I2C_CLK_MAX = 400 kHz) - Effective Master Clock depends on Host Platform.

Establish a connection via JCShell:
 /term Remote|<ip-address>:<port>
 e.g.
 /term Remote|192.168.1.27:8050

Server: waiting for connections...
```

For a client to connect to the server it needs to know the IP address of the server. In the following example the Configure Tool - built with the JRCP\_V1 SMC layer - runs on a Linux PC and connects to the JRCP\_v1 server running on the embedded platform (with IP address 192.168.1.100).

```
user@dell-linux:~/$./A71CHConfigTool 192.168.2.81:8050 interactive
a71chConfig (Rev 1.20) .. connect to A71CH. Chunksize at link layer = 256.
PlugAndTrust_HostLib_v02.12.00_20191210
Applet Version : 0x131X
SecureBox Version: 0x0X

=====SELECT-DONE=====
HostLib Version : 0x020C
Applet-Rev:SecureBox-Rev : 0x0131:0x0000
>>> info device
A71CH in Debug Mode Version (SCP03 is not set up)
selectResponse: 0x0131
transportLockState: 0x03 (Transport Lock NOT YET set)
injectLockState: 0x02 (Unlocked)
gpStorageSize: 4096
uid (LEN=18):
47:90:70:02:47:91:12:10:80:04:00:88:04:98:90:53:48:12
certUid (LEN=10):
70:02:80:04:00:88:04:98:90:53
```

## 9.11 Using own Platform SCP03 Keys

The Plug & Trust MW can use PlatformSCP03 keys from file system. The key files for different platforms are defined as:

### For Android

```
#define EX_SSS_SCP03_FILE_DIR "/data/vendor/SE05x/"
#define EX_SSS_SCP03_FILE_PATH EX_SSS_SCP03_FILE_DIR "plain_scp.txt"
```

### For Linux

```
#define EX_SSS_SCP03_FILE_DIR "/tmp/SE05X/"
#define EX_SSS_SCP03_FILE_PATH EX_SSS_SCP03_FILE_DIR "plain_scp.txt"
```

### For Windows

```
#define EX_SSS_SCP03_FILE_DIR "C:\\nxp\\SE05X\\"
#define EX_SSS_SCP03_FILE_PATH EX_SSS_SCP03_FILE_DIR "plain_scp.txt"
```

You need to create a file at this location to allow the MW to pick up the file automatically. Another option is to set the environment variable `EX_SSS_BOOT_SCP03_PATH` to the complete file path.

The MW will first look for the file at the above path, if it is not able to find the file, it will try to use the environment variable, and lastly, it will fall back to pre-compiled keys.

---

**Note:** For Android systems, it is important to update sepolicy to allow access to Platform SCP03 keys directory. Refer to AOSP setup [Section 9.5.9 AOSP build Environment for Hikey960](#) for details on required system patches.

---

It is advisable to create a file at this location to allow MW to use those keys instead of pre-compiled keys as the user can also rotate the keys in which case if the MW was using pre-compiled keys, all further operations will fail.

An example of file format is:

```
This is a comment, empty lines and comment lines allowed.
ENC 35C29245895EA34F6136155F8209D6CD # Trailing comment
MAC AF172D5D54F7C0D5C10A05B9F1207F78 # Optional trailing comment
DEK A2BC8438BF77015B361A4425F239FA29 # Optional trailing comment
```

Replace the ENC, MAC and DEK keys with your own keys.

For information on rotating Platform SCP03 keys, refer to [Section 5.19 SE05X Rotate PlatformSCP Keys Demo](#)

## 9.12 Write APDU to buffer

See [Section 5.38 Write APDU to buffer](#).

## 9.13 Plug & Trust MW APIs

### 9.13.1 Class Hierarchy

- *Struct NXECKey03\_StaticCtx\_t*
- *Struct NXSCP03\_AuthCtx\_t*
- *Struct NXSCP03\_DynCtx\_t*
- *Struct NXSCP03\_StaticCtx\_t*
- *Struct SE05x\_Applet\_Feature\_t*
- *Struct SE05x\_AuthCtx\_ECKey\_t*
- *Struct SE05x\_AuthCtx\_ID\_t*
- *Struct SE05x\_I2CM\_cmd\_t*
- *Struct SE05x\_I2CM\_configData\_t*
- *Struct SE05x\_I2CM\_readData\_t*
- *Struct SE05x\_I2CM\_securityData\_t*



- *Struct SE05x\_I2CM\_structuralIssue\_t*
- *Struct SE05x\_I2CM\_writeData\_t*
- *Struct SE\_AuthCtx\_t*
- *Struct SE\_Connect\_Ctx\_t*
- *Struct SM\_SECURE\_SCP03\_KEYOBJ*
- *Struct sss\_aead\_t*
- *Struct sss\_asymmetric\_t*
- *Struct sss\_connect\_ctx\_t*
- *Struct sss\_derive\_key\_t*
- *Struct sss\_digest\_t*
- *Struct sss\_ecc\_point\_t*
- *Struct sss\_eccgfp\_group\_t*
- *Struct sss\_key\_store\_t*
- *Struct sss\_mac\_t*
- *Struct sss\_object\_t*
- *Struct sss\_policy\_asym\_key\_u*
- *Struct sss\_policy\_common\_pcr\_value\_u*
- *Struct sss\_policy\_common\_u*
- *Struct sss\_policy\_counter\_u*
- *Struct sss\_policy\_file\_u*
- *Struct sss\_policy\_pcr\_u*
- *Struct sss\_policy\_session\_u*
- *Struct sss\_policy\_sym\_key\_u*
- *Struct sss\_policy\_t*
- *Struct sss\_policy\_u*
- *Struct sss\_policy\_userid\_u*
- *Struct sss\_rng\_context\_t*
- *Struct sss\_se05x\_aead\_t*
- *Struct sss\_se05x\_asymmetric\_t*
- *Struct sss\_se05x\_attst\_comp\_data\_t*
- *Struct sss\_se05x\_attst\_data\_t*
- *Struct sss\_se05x\_derive\_key\_t*
- *Struct sss\_se05x\_digest\_t*
- *Struct sss\_se05x\_key\_store\_t*
- *Struct sss\_se05x\_mac\_t*
- *Struct sss\_se05x\_object\_t*

- *Struct sss\_se05x\_rng\_context\_t*
- *Struct sss\_se05x\_session\_t*
- *Struct sss\_se05x\_symmetric\_t*
- *Struct sss\_se05x\_tunnel\_context\_t*
- *Struct sss\_session\_t*
- *Struct sss\_symmetric\_t*
- *Struct sss\_tunnel\_t*
- *Struct tlvHeader\_t*
- *Enum SE05x\_AeadAlgo\_t*
- *Enum SE05x\_AppletConfig\_t*
- *Enum SE05x\_AppletResID\_t*
- *Enum SE05x\_AttestationAlgo\_t*
- *Enum SE05x\_AttestationType\_t*
- *Enum SE05x\_Cipher\_Oper\_OneShot\_t*
- *Enum SE05x\_Cipher\_Oper\_t*
- *Enum SE05x\_CipherMode\_t*
- *Enum SE05x\_CryptoContext\_t*
- *Enum SE05x\_CryptoObject\_t*
- *Enum SE05x\_DigestMode\_t*
- *Enum SE05x\_ECCurve\_t*
- *Enum SE05x\_ECCurveParam\_t*
- *Enum SE05x\_ECDAASignatureAlgo\_t*
- *Enum SE05x\_ECSignatureAlgo\_t*
- *Enum SE05x\_EDSignatureAlgo\_t*
- *Enum SE05x\_HealthCheckMode\_t*
- *Enum SE05x\_HkdfMode\_t*
- *Enum SE05x\_I2CM\_Baud\_Rate\_t*
- *Enum SE05x\_I2CM\_securityReq\_t*
- *Enum SE05x\_I2CM\_status\_t*
- *Enum SE05x\_I2CM\_TAG\_t*
- *Enum SE05x\_I2CM\_TLV\_type\_t*
- *Enum SE05x\_INS\_t*
- *Enum SE05x\_KeyPart\_t*
- *Enum SE05x\_LockIndicator\_t*
- *Enum SE05x\_LockState\_t*
- *Enum SE05x\_Mac\_Oper\_t*

- *Enum SE05x\_MACAlgo\_t*
- *Enum SE05x\_MemoryType\_t*
- *Enum SE05x\_MemTyp\_t*
- *Enum SE05x\_MoreIndicator\_t*
- *Enum SE05x\_Origin\_t*
- *Enum SE05x\_P1\_t*
- *Enum SE05x\_P2\_t*
- *Enum SE05x\_PlatformSCPRequest\_t*
- *Enum SE05x\_RestrictMode\_t*
- *Enum SE05x\_Result\_t*
- *Enum SE05x\_RSABitLength\_t*
- *Enum SE05x\_RSAEncryptionAlgo\_t*
- *Enum SE05x\_RSAKeyComponent\_t*
- *Enum SE05x\_RSAKeyFormat\_t*
- *Enum SE05x\_RSAPubKeyComp\_t*
- *Enum SE05x\_RSASignAlgo\_t*
- *Enum SE05x\_RSASignatureAlgo\_t*
- *Enum SE05x\_SecObjTyp\_t*
- *Enum SE05x\_SetIndicator\_t*
- *Enum SE05x\_SW12\_t*
- *Enum SE05x\_SymmKeyType\_t*
- *Enum SE05x\_TAG\_t*
- *Enum SE05x\_TLSPerformPRFType\_t*
- *Enum SE05x\_TransientIndicator\_t*
- *Enum SE05x\_TransientType\_t*
- *Enum SE\_AuthType\_t*
- *Enum sss\_access\_permission\_t*
- *Enum sss\_algorithm\_t*
- *Enum sss\_cipher\_type\_t*
- *Enum sss\_connection\_type\_t*
- *Enum sss\_key\_object\_mode\_t*
- *Enum sss\_key\_part\_t*
- *Enum sss\_key\_store\_prop\_au8\_t*
- *Enum sss\_mode\_t*
- *Enum sss\_policy\_type\_u*
- *Enum sss\_s05x\_sesion\_prop\_au8\_t*

- *Enum sss\_s05x\_session\_prop\_u32\_t*
- *Enum sss\_session\_prop\_au8\_t*
- *Enum sss\_session\_prop\_u32\_t*
- *Enum sss\_status\_t*
- *Enum sss\_tunnel\_dest\_t*
- *Enum sss\_type\_t*
- *Union SE05x\_CryptoModeSubType\_t*
- *Union SE05x\_I2CM\_INS\_type\_t*

### 9.13.2 File Hierarchy

- **dir\_doc**
  - **dir\_doc\_input**
    - file\_doc\_input\_modules.h
    - file\_doc\_input\_sss\_object.h
- **dir\_hostlib**
  - **dir\_hostlib\_hostLib**
    - **dir\_hostlib\_hostLib\_inc**
      - file\_hostlib\_hostLib\_inc\_nxEnsure.h
      - file\_hostlib\_hostLib\_inc\_nxScp03\_Types.h
      - file\_hostlib\_hostLib\_inc\_se05x\_enums.h
    - **dir\_hostlib\_hostLib\_se05x\_03\_xx\_xx**
      - file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU.h
      - file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h
- **dir\_sss**
  - **dir\_sss\_doc**
    - file\_sss\_doc\_modules.h
  - **dir\_sss\_ex**
    - **dir\_sss\_ex\_inc**
      - file\_sss\_ex\_inc\_ex\_sss\_main\_inc.h
  - **dir\_sss\_inc**
    - file\_sss\_inc\_fsl\_sss\_api.h
    - file\_sss\_inc\_fsl\_sss\_policy.h
    - file\_sss\_inc\_fsl\_sss\_se05x\_apis.h
    - file\_sss\_inc\_fsl\_sss\_se05x\_types.h
  - **dir\_sss\_plugin**
    - **dir\_sss\_plugin\_mbedtls**

- file\_sss\_plugin\_mbedtls\_sss\_mbedtls.h

### 9.13.3 Full API

#### Classes and Structs

##### Struct NXECKKey03\_StaticCtx\_t

- Defined in file\_hostlib\_hostLib\_inc\_nxScp03\_Types.h

#### Struct Documentation

**struct NXECKKey03\_StaticCtx\_t**  
Static part of keys for FAST SCP

##### Public Members

*sss\_object\_t* **HostEcdsaObj**  
Host ECDSA Private key

*sss\_object\_t* **HostEcKeypair**  
Host ephemeral ECC key pair

*sss\_object\_t* **masterSec**  
Host master Secret

*sss\_object\_t* **SeEcPubKey**  
SE ECC public key

##### Struct NXSCP03\_AuthCtx\_t

- Defined in file\_hostlib\_hostLib\_inc\_nxScp03\_Types.h

#### Struct Documentation

**struct NXSCP03\_AuthCtx\_t**  
Static and Dynamic Context in one Context.  
Depending on system, these objects may point to keys inside other security system.

##### Public Members

*NXSCP03\_DynCtx\_t* \***pDyn\_ctx**  
session keys data

*NXSCP03\_StaticCtx\_t* \***pStatic\_ctx**  
.static keys data

## Struct NXSCP03\_DynCtx\_t

- Defined in file\_hostlib\_hostLib\_inc\_nxScp03\_Types.h

### Struct Documentation

**struct NXSCP03\_DynCtx\_t**

Dynamic SCP03 Context.

This structure is filled **after** establishing an SCP03 session.

#### Public Members

*SE\_AuthType\_t* **authType**

Handle differnt types of auth.. PlatformSCP / AppletSCP

uint8\_t **cCounter**[16]

command counter

*sss\_object\_t* **Enc**

session channel encryption key

*sss\_object\_t* **Mac**

session command authentication key

uint8\_t **MCV**[16]

MAC chaining value.

*sss\_object\_t* **Rmac**

session response authentication key

uint8\_t **SecurityLevel**

security level set

## Struct NXSCP03\_StaticCtx\_t

- Defined in file\_hostlib\_hostLib\_inc\_nxScp03\_Types.h

### Struct Documentation

**struct NXSCP03\_StaticCtx\_t**

Static SCP03 Context.

This structure is filled **before** establishing an SCP03 session.

Depending on system, these objects may point to keys inside other security system.

## Public Members

*sss\_object\_t* **Dek**  
data encryption key obj

*sss\_object\_t* **Enc**  
Encryption key object

uint8\_t **keyVerNo**  
Key version no to use for chanel authentication in SCP03

*sss\_object\_t* **Mac**  
static secure channel authentication key obj

## Struct SE05x\_Applet\_Feature\_t

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_types.h

## Struct Documentation

**struct SE05x\_Applet\_Feature\_t**  
Used to disable Applet Features via sss\_se05x\_set\_feature

## Public Members

uint8\_t **AppletConfig\_AES**  
Writing AESKey objects

uint8\_t **AppletConfig\_DES**  
Writing DESKey objects

uint8\_t **AppletConfig\_DH\_MONT**  
Use of curve RESERVED\_ID\_ECC\_MONT\_DH\_25519

uint8\_t **AppletConfig\_ECDAA**  
Use of curve TPM\_ECC\_BN\_P256

uint8\_t **AppletConfig\_ECDSA\_ECDH\_ECDHE**  
EC DSA and DH support

uint8\_t **AppletConfig\_EDDSA**  
Use of curve RESERVED\_ID\_ECC\_ED\_25519

uint8\_t **AppletConfig\_HMAC**  
Writing HMACKey objects

uint8\_t **AppletConfig\_I2CM**  
I2C Master support (see 4.17) in APDU Spec

uint8\_t **AppletConfig\_MIFARE**  
Mifare DESFire support (see 4.15) in APDU Spec

uint8\_t **AppletConfig\_PBKDF**  
PBKDF2

uint8\_t **AppletConfig\_RFU1**  
Allocated value undefined and reserved for future use

uint8\_t **AppletConfig\_RFU21**  
RFU

uint8\_t **AppletConfig\_RSA\_CRT**  
Writing RSAKey objects

uint8\_t **AppletConfig\_RSA\_PLAIN**  
Writing RSAKey objects

uint8\_t **AppletConfig\_TLS**  
TLS Handshake support commands (see 4.16) in APDU Spec

### Struct SE05x\_AuthCtx\_ECKey\_t

- Defined in file\_hostlib\_hostLib\_inc\_nxScp03\_Types.h

### Struct Documentation

**struct SE05x\_AuthCtx\_ECKey\_t**  
Keys to connect for a ECKey Connection

#### Public Members

*NXSCP03\_DynCtx\_t* \***pDyn\_ctx**  
The Dynamic part of the ECKey Authentication  
We derive/compute the session keys based on the pStatic\_ctx.

*NXECKey03\_StaticCtx\_t* \***pStatic\_ctx**  
The Input/Static part of the ECKey Authentication  
We start/initiate a session with the keys here.

### Struct SE05x\_AuthCtx\_ID\_t

- Defined in file\_hostlib\_hostLib\_inc\_nxScp03\_Types.h

### Struct Documentation

**struct SE05x\_AuthCtx\_ID\_t**  
UserID / PIN baed authentication object  
This is required to open an UserID / PIN based session to the SE.



### Public Members

*sss\_object\_t* \*pObj

The corresponding authentication object on the Host

### Struct SE05x\_I2CM\_cmd\_t

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_types.h

### Struct Documentation

**struct SE05x\_I2CM\_cmd\_t**

Individual entry in array of TLV commands, with type

*Se05x\_i2c\_master\_txn* would expect an array of these.

### Public Members

*SE05x\_I2CM\_INS\_type\_t* cmd

Individual entry in array of TLV commands.

*SE05x\_I2CM\_TLV\_type\_t* type

### Struct SE05x\_I2CM\_configData\_t

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_types.h

### Struct Documentation

**struct SE05x\_I2CM\_configData\_t**

Data Configuration for I2CM

### Public Members

uint8\_t I2C\_addr

7 Bit address of I2C slave

*SE05x\_I2CM\_Baud\_Rate\_t* I2C\_baudRate

What baud rate

*SE05x\_I2CM\_status\_t* status

return status of the config operation

## Struct SE05x\_I2CM\_readData\_t

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_types.h

### Struct Documentation

**struct SE05x\_I2CM\_readData\_t**  
Read to I2CM from I2C Slave

#### Public Members

uint8\_t \***rdBuf**  
Output. rdBuf will point to Host buffer.

*SE05x\_I2CM\_status\_t* **rdStatus**  
[Out] status of the operation

uint16\_t **readLength**  
How many bytes to read

## Struct SE05x\_I2CM\_securityData\_t

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_types.h

### Struct Documentation

**struct SE05x\_I2CM\_securityData\_t**  
Security Configuration for I2CM.

#### Public Members

uint32\_t **keyObject**  
object used for the operation

*SE05x\_I2CM\_securityReq\_t* **operation**  
Additional operation on data read by I2C

## Struct SE05x\_I2CM\_structuralIssue\_t

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_types.h

## Struct Documentation

**struct SE05x\_I2CM\_structuralIssue\_t**

Used to report error response, not for outgoing command

### Public Members

*SE05x\_I2CM\_status\_t* **issueStatus**

[Out] In case there is any structural issue

## Struct SE05x\_I2CM\_writeData\_t

- Defined in file\_sss\_inc\_fsl\_oss\_se05x\_types.h

## Struct Documentation

**struct SE05x\_I2CM\_writeData\_t**

Write From I2CM to I2C Slave.

### Public Members

uint8\_t **\*writebuf**

Buffer to be written

uint8\_t **writeLength**

How many bytes to write

*SE05x\_I2CM\_status\_t* **wrStatus**

[Out] status of the operation

## Struct SE\_AuthCtx\_t

- Defined in file\_hostlib\_hostLib\_inc\_nxScp03\_Types.h

## Struct Documentation

**struct SE\_AuthCtx\_t**

Authentication mechanisms

### Public Members

*SM\_SECURE\_SCP03\_KEYOBJ* **a71chAuthKeys**

Legacy, only for A71CH with Host Crypto

*SE\_AuthType\_t* **authType**

How exactly we are going to authenticate of the system.

Since `ctx` is a union, this is needed to know exactly how we are going to authenticate.

**union *SE\_AuthCtx\_t*::[anonymous] ctx**

Depending on *authType*, the input and output parameters.

This has both input and output parameters.

Input is for Keys that are used to initiate the connection. While connecting, session keys/parameters are generated and they are also part of this context.

In any case, we connect to only one type

**uint8\_t data[SSS\_AUTH\_MAX\_CONTEXT\_SIZE]**

***SE05x\_AuthCtx\_ECKey\_t* eckey**

For ECKey

**struct *SE\_AuthCtx\_t*::[anonymous]::[anonymous] extension**

Reserved memory for implementation specific extension

***SE05x\_AuthCtx\_ID\_t* idobj**

For UserID/PIN based Authentication

***NXSCP03\_AuthCtx\_t* scp03**

For PlatformSCP / Applet SCP.

Same SCP context will be used for platform and applet scp03

## Struct *SE\_Connect\_Ctx\_t*

- Defined in file *\_hostlib\_hostLib\_inc\_nxScp03\_Types.h*

## Struct Documentation

**struct *SE\_Connect\_Ctx\_t***

When connecting to a secure element,

Extension of *sss\_connect\_ctx\_t*

## Public Members

***SE\_AuthCtx\_t* auth**

If we need to authenticate, add required objects for authentication

**SSS\_Conn\_Type\_t connType**

How exactly are we going to connect physically

**U32 i2cAddress**

I2C address on embedded devices.

**const char \*portName**

Connection port name for Socket names, etc.

**uint8\_t refresh\_session**

If we need to refresh session, SE050 specific

***sss\_policy\_session\_u* \*session\_policy**

If some policy restrictions apply when we connect, point it here

**uint16\_t sizeofStructure**

to support binary compatibility/check, sizeofStructure helps

`uint8_t skip_select_applet`

In the case of Key Rotation, and other use cases where we do not select the IoT Applet and skip the selection of the IoT Applet.

One of the use cases is to do platform SCP key rotation.

When set to 0: Do not skip IoT Applet selection and run as-is.

When set to 1: Skip selection of card manager. Skip selection of Applet.

Internally, if there is platform SCP selected as Auth mechanism during compile time, the internal logic would Select the card manager. But, skip selection of the Applet.

*sss\_tunnel\_t* \*`tunnelCtx`

If we connect logically, via some software layer

## Struct SM\_SECURE\_SCP03\_KEYOBJ

- Defined in file\_hostlib\_hostLib\_inc\_nxScp03\_Types.h

## Struct Documentation

**struct SM\_SECURE\_SCP03\_KEYOBJ**

Legacy, only for A71CH with Host Crypto

### Public Members

*sss\_object\_t* **pKeyDek**

SSS AES Dek Key object.

*sss\_object\_t* **pKeyEnc**

SSS AES Enc Key object.

*sss\_object\_t* **pKeyMac**

SSS AES Mac Key object.

## Struct sss\_aead\_t

- Defined in file\_sss\_inc\_fsl\_sss\_api.h

## Struct Documentation

**struct sss\_aead\_t**

Authenticated Encryption with Additional Data.

## Public Members

*sss\_algorithm\_t* **algorithm**

TODO : Algorithm to be applied

uint8\_t **data**[(0 + (2 \* sizeof(void \*)) + (2 \* sizeof(int)) + (2 \* sizeof(void \*)) + 16)]

**struct** *sss\_aead\_t*::**[anonymous] extension**

Reserved memory for implementation specific extension

*sss\_object\_t* \***keyObject**

Key to be used for asymmetric

*sss\_mode\_t* **mode**

TODO : High level operation

*sss\_session\_t* \***session**

Virtual connection between application (user context) and specific security subsystem and function thereof.

## Struct sss\_asymmetric\_t

- Defined in file\_sss\_inc\_fsl\_sss\_api.h

## Struct Documentation

**struct** *sss\_asymmetric\_t*

Asymmetric Cryptographic operations.

e.g. RSA/ECC.

## Public Members

*sss\_algorithm\_t* **algorithm**

Algorithm to be applied, e.g. ECDSA

uint8\_t **data**[(0 + (2 \* sizeof(void \*)) + (3 \* sizeof(int)) + (2 \* sizeof(void \*)) + 16)]

**struct** *sss\_asymmetric\_t*::**[anonymous] extension**

Reserved memory for implementation specific extension

*sss\_object\_t* \***keyObject**

KeyObject used for Asymmetric operation

*sss\_mode\_t* **mode**

Mode of operation for the Asymmetric operation. e.g. Sign/Verify/Encrypt/Decrypt

*sss\_session\_t* \***session**

Pointer to root session

## Struct `sss_connect_ctx_t`

- Defined in file `hostlib_hostLib_inc_nxScp03_Types.h`

### Struct Documentation

**struct `sss_connect_ctx_t`**

Wrapper structure `sss_connect_ctx_t`

#### Public Members

*SE\_AuthCtx\_t* **auth**

If we need to authenticate, add required objects for authentication

`uint8_t data[SSS_CONNECT_MAX_CONTEXT_SIZE]`

**struct `sss_connect_ctx_t::[anonymous] extension`**

Reserved memory for implementation specific extension

*sss\_policy\_session\_u* \***session\_policy**

If some policy restrictions apply when we connect, point it here

`uint16_t sizeofStructure`

To support binary compatibility/check, sizeofStructure helps

## Struct `sss_derive_key_t`

- Defined in file `sss_inc_fsl_sss_api.h`

### Struct Documentation

**struct `sss_derive_key_t`**

Key derivation

#### Public Members

*sss\_algorithm\_t* **algorithm**

Algorithm to be applied, e.g. ...

`uint8_t data[(0 + (2 * sizeof(void *))) + (2 * sizeof(int)) + (2 * sizeof(void *)) + 16]`

**struct `sss_derive_key_t::[anonymous] extension`**

Reserved memory for implementation specific extension

*sss\_object\_t* \***keyObject**

KeyObject used to derive key s

*sss\_mode\_t* **mode**

Mode of operation for .... e.g. ...

*sss\_session\_t* \***session**

Pointer to the session

## Struct `sss_digest_t`

- Defined in file `_sss_inc_fsl_sss_api.h`

### Struct Documentation

**struct `sss_digest_t`**  
Message Digest operations

#### Public Members

*`sss_algorithm_t`* **algorithm**

Algorithm to be applied, e.g SHA1, SHA256

`uint8_t data[(0 + (1 * sizeof(void *))) + (3 * sizeof(int)) + (2 * sizeof(void *)) + 16]`

`size_t digestFullLen`

Full digest length per algorithm definition. This field is initialized along with algorithm.

**struct *`sss_digest_t`*::[anonymous] extension**

Reserved memory for implementation specific extension

*`sss_mode_t`* **mode**

Mode of operation, e.g Sign/Verify

*`sss_session_t`* \***session**

Virtual connection between application (user context) and specific security subsystem and function thereof.

## Struct `sss_ecc_point_t`

- Defined in file `_sss_inc_fsl_sss_api.h`

### Struct Documentation

**struct `sss_ecc_point_t`**  
XY Co-ordinates for ECC Curves

#### Public Members

`uint8_t *X`

X Point

`uint8_t *Y`

Y Point



## Struct `sss_eccgfp_group_t`

- Defined in file `_sss_inc_fsl_sss_api.h`

### Struct Documentation

**struct `sss_eccgfp_group_t`**  
ECC Curve Parameter

#### Public Members

`uint8_t *a`  
ECC parameter a

`uint8_t *b`  
ECC parameter b

`sss_ecc_point_t *G`  
ECC parameter G

`uint8_t *h`  
ECC parameter h

`uint8_t *n`  
ECC parameter n

`uint8_t *p`  
ECC parameter P

## Struct `sss_key_store_t`

- Defined in file `_sss_inc_fsl_sss_api.h`

### Struct Documentation

**struct `sss_key_store_t`**  
Store for secure and non secure key objects within a cryptographic system.

- A cryptographic system may have more than partitions to store such keys.

#### Public Members

`uint8_t data[(0 + (1 * sizeof(void *))) + (4 * sizeof(void *)) + 16]`

**struct `sss_key_store_t::[anonymous] extension`**  
Reserved memory for implementation specific extension

`sss_session_t *session`  
Virtual connection between application (user context) and specific security subsystem and function thereof.

## Struct `sss_mac_t`

- Defined in `file_sss_inc_fsl_sss_api.h`

## Struct Documentation

### **struct `sss_mac_t`**

Message Authentication Code.

#### **Public Members**

*sss\_algorithm\_t* **algorithm**

Algorithm to be applied, e.g. MAC/CMAC

`uint8_t data[(0 + (2 * sizeof(void *))) + (2 * sizeof(int)) + (2 * sizeof(void *)) + 32]`

**struct *sss\_mac\_t*::[anonymous] extension**

Reserved memory for implementation specific extension

*sss\_object\_t* \***keyObject**

Key to be used for ...

*sss\_mode\_t* **mode**

Mode of operation for MAC e.g. ...

*sss\_session\_t* \***session**

Virtual connection between application (user context) and specific security subsystem and function thereof.

## Struct `sss_object_t`

- Defined in `file_sss_inc_fsl_sss_api.h`

## Struct Documentation

### **struct `sss_object_t`**

An object (secure / non-secure) within a Key Store.

#### **Public Members**

`uint32_t` **cipherType**

cipherType type from *sss\_cipher\_type\_t*

`uint8_t data[(0 + (1 * sizeof(void *))) + (2 * sizeof(int)) + (4 * sizeof(void *)) + 16]`

**struct *sss\_object\_t*::[anonymous] extension**

Reserved memory for implementation specific extension

`uint32_t` **keyId**

Application specific key identifier. The keyId is kept in the key store along with the key data and other properties.

*sss\_key\_store\_t* \***keyStore**

key store holding the data and other properties

uint32\_t **objectType**

The type/part of object is refernced from *sss\_key\_part\_t*

## Struct sss\_policy\_asym\_key\_u

- Defined in file\_sss\_inc\_fsl\_sss\_policy.h

## Struct Documentation

**struct sss\_policy\_asym\_key\_u**

Policies applicable to Asymmetric KEY

### Public Members

uint8\_t **can\_Attest**

Allow to attest an object

uint8\_t **can\_Decrypt**

Allow decryption

uint8\_t **can\_Encrypt**

Allow encryption

uint8\_t **can\_Gen**

Allow to (re)generate the object

uint8\_t **can\_Import\_Export**

Allow to imported or exported

uint8\_t **can\_KA**

Allow key agreement

uint8\_t **can\_KD**

Allow key derivation

uint8\_t **can\_Read**

Allow to read the object

uint8\_t **can\_Sign**

Allow signature generation

uint8\_t **can\_Verify**

Allow signature verification

uint8\_t **can\_Wrap**

Allow key wrapping

uint8\_t **can\_Write**

Allow to write the object

uint8\_t **forbid\_Derived\_Output**

Forbid derived output

### Struct `sss_policy_common_pcr_value_u`

- Defined in `file_sss_inc_fsl_sss_policy.h`

#### Struct Documentation

**struct `sss_policy_common_pcr_value_u`**  
Common PCR Value Policies for all object types

##### Public Members

`uint8_t pcrExpectedValue[32]`  
Expected value of the PCR

`uint32_t pcrObjId`  
PCR object ID

### Struct `sss_policy_common_u`

- Defined in `file_sss_inc_fsl_sss_policy.h`

#### Struct Documentation

**struct `sss_policy_common_u`**  
Common Policies for all object types

##### Public Members

`uint8_t can_Delete`  
Allow to delete the object

`uint8_t forbid_All`  
Forbid all operations

`uint8_t req_Sm`  
Require having secure messaging enabled with encryption and integrity on the command

### Struct `sss_policy_counter_u`

- Defined in `file_sss_inc_fsl_sss_policy.h`

## Struct Documentation

### **struct sss\_policy\_counter\_u**

All policies related to secure object type Counter

#### **Public Members**

uint8\_t **can\_Read**

Allow to read the object

uint8\_t **can\_Write**

Allow to write the object

### **Struct sss\_policy\_file\_u**

- Defined in file\_sss\_inc\_fsl\_sss\_policy.h

## Struct Documentation

### **struct sss\_policy\_file\_u**

All policies related to secure object type File

#### **Public Members**

uint8\_t **can\_Read**

Allow to read the object

uint8\_t **can\_Write**

Allow to write the object

### **Struct sss\_policy\_pcr\_u**

- Defined in file\_sss\_inc\_fsl\_sss\_policy.h

## Struct Documentation

### **struct sss\_policy\_pcr\_u**

All policies related to secure object type PCR

## Public Members

`uint8_t can_Read`  
Allow to read the object

`uint8_t can_Write`  
Allow to write the object

## Struct `sss_policy_session_u`

- Defined in `file_sss_inc_fsl_sss_policy.h`

## Struct Documentation

**struct `sss_policy_session_u`**  
Policy applicable to a session

## Public Members

`uint8_t allowRefresh`  
Whether this session can be refreshed without losing context. And also reset `maxDurationOfSession_sec` / `maxOperationsInSession`

`uint8_t has_MaxDurationOfSession_sec`  
Whether `maxOperationsInSession` is set. This is to ensure '0 == `maxDurationOfSession_sec`' does not get set by middleware.

`uint8_t has_MaxOperationsInSession`  
Whether `maxOperationsInSession` is set. This is to ensure '0 == `maxOperationsInSession`' does not get set by middleware.

`uint16_t maxDurationOfSession_sec`  
Session can be used for this much time, in seconds

`uint16_t maxOperationsInSession`  
Number of operations permitted in a session

## Struct `sss_policy_sym_key_u`

- Defined in `file_sss_inc_fsl_sss_policy.h`

## Struct Documentation

**struct `sss_policy_sym_key_u`**  
Policies applicable to Symmetric KEY

## Public Members

`uint8_t can_Decrypt`  
Allow decryption

`uint8_t can_Desfire_Auth`  
Allow to perform DESFire authentication

`uint8_t can_Desfire_Dump`  
Allow to dump DESFire session keys

`uint8_t can_Encrypt`  
Allow encryption

`uint8_t can_Gen`  
Allow to (re)generate the object

`uint8_t can_Import_Export`  
Allow to imported or exported

`uint8_t can_KD`  
Allow key derivation

`uint8_t can_Sign`  
Allow signature generation

`uint8_t can_Verify`  
Allow signature verification

`uint8_t can_Wrap`  
Allow key wrapping

`uint8_t can_Write`  
Allow to write the object

`uint8_t forbid_Derived_Output`  
Forbid derived output

## Struct `sss_policy_t`

- Defined in file `_sss_inc_fsl_sss_policy.h`

## Struct Documentation

**struct `sss_policy_t`**  
An array of policies *`sss_policy_u`*

## Public Members

`size_t nPolicies`

Number of policies

`const sss_policy_u *policies[(10)]`

Array of unique policies, this needs to be allocated based nPolicies

## Struct `sss_policy_u`

- Defined in `file_sss_inc_fsl_sss_policy.h`

## Struct Documentation

`struct sss_policy_u`

Unique/individual policy. For any operation, you need array of *sss\_policy\_u*.

## Public Members

`sss_policy_asym_key_u asymmkey`

`uint32_t auth_obj_id`

Auth ID for each Object Policy, invalid for session policy type == KPolicy\_Session

`sss_policy_common_u common`

`sss_policy_common_pcr_value_u common_pcr_value`

`sss_policy_counter_u counter`

`sss_policy_file_u file`

`sss_policy_pcr_u pcr`

`sss_policy_userid_u pin`

`union sss_policy_u::[anonymous] policy`

Union of applicable policies based on the type of object

`sss_policy_session_u session`

`sss_policy_sym_key_u symmkey`

`sss_policy_type_u type`

Secure Object Type

## Struct `sss_policy_userid_u`

- Defined in `file_sss_inc_fsl_sss_policy.h`



## Struct Documentation

### **struct sss\_policy\_userid\_u**

All policies related to secure object type UserID

#### **Public Members**

uint8\_t **can\_Write**

Allow to write the object

## Struct sss\_rng\_context\_t

- Defined in file\_sss\_inc\_fsl\_sss\_api.h

## Struct Documentation

### **struct sss\_rng\_context\_t**

Random number generator context

#### **Public Members**

**struct sss\_rng\_context\_t::[anonymous] context**

Reserved memory for implementation specific extension

uint8\_t **data**[(0 + (1 \* sizeof(void \*)) + (2 \* sizeof(void \*)) + 16)]

*sss\_session\_t* \***session**

Pointer to the session

## Struct sss\_se05x\_aead\_t

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_types.h

## Struct Documentation

### **struct sss\_se05x\_aead\_t**

Authenticated Encryption with Additional Data.

#### **Public Members**

*sss\_algorithm\_t* **algorithm**

TODO : Algorithm to be applied

uint8\_t **cache\_data**[16]

Cache in case of un-aligned inputs

size\_t **cache\_data\_len**

How much we have cached

`SE05x_CryptoObjectID_t cryptoObjectId`

Implementation specific part

`sss_se05x_object_t *keyObject`

Key to be used for asymmetric

`sss_mode_t mode`

TODO : High level operation

`sss_se05x_session_t *session`

Virtual connection between application (user context) and specific security subsystem and function thereof.

## Struct `sss_se05x_asymmetric_t`

- Defined in file `_sss_inc_fsl_sss_se05x_types.h`

## Struct Documentation

**struct `sss_se05x_asymmetric_t`**

Asymmetric Cryptographic operations.

e.g. RSA/ECC.

### Public Members

`sss_algorithm_t algorithm`

Algorithm to be applied, e.g. ECDSA

`sss_se05x_object_t *keyObject`

KeyObject used for Asymmetric operation

`sss_mode_t mode`

Mode of operation for the Asymmetric operation. e.g. Sign/Verify/Encrypt/Decrypt

`sss_se05x_session_t *session`

Pointer to root session

## Struct `sss_se05x_attst_comp_data_t`

- Defined in file `_sss_inc_fsl_sss_se05x_types.h`

## Struct Documentation

**struct `sss_se05x_attst_comp_data_t`**

Attestation data

## Public Members

`uint8_t attribute[MAX_POLICY_BUFFER_SIZE + 15]`  
Attributes

`size_t attributeLen`  
Length of Attribute

`uint8_t chipId[SE050_MODULE_UNIQUE_ID_LEN]`  
Unique ID of SE050

`size_t chipIdLen`  
Length of the Unique ID

`uint8_t outrandom[16]`  
Random used during attestation

`size_t outrandomLen`  
length of outrandom

`uint8_t signature[256]`  
Signature for attestation

`size_t signatureLen`  
Length of signature

`SE05x_TimeStamp_t timeStamp`  
time stamp

`size_t timeStampLen`  
Length of timeStamp

## Struct `sss_se05x_attst_data_t`

- Defined in file `_sss_inc_fsl_sss_se05x_types.h`

## Struct Documentation

`struct sss_se05x_attst_data_t`  
Data to be read with attestation

## Public Members

`sss_se05x_attst_comp_data_t data[SE05X_MAX_ATTST_DATA]`  
While reading RSA Objects, modulus and public exponent get attested separately,

`uint8_t valid_number`  
How many entries to attest

## Struct `sss_se05x_derive_key_t`

- Defined in file `_sss_inc_fsl_sss_se05x_types.h`

### Struct Documentation

**struct `sss_se05x_derive_key_t`**  
Key derivation

#### Public Members

*sss\_algorithm\_t* **algorithm**  
Algorithm to be applied, e.g. ...

*sss\_se05x\_object\_t* \***keyObject**  
KeyObject used to derive key s

*sss\_mode\_t* **mode**  
Mode of operation for .... e.g. ...

*sss\_se05x\_session\_t* \***session**  
Pointer to the session

## Struct `sss_se05x_digest_t`

- Defined in file `_sss_inc_fsl_sss_se05x_types.h`

### Struct Documentation

**struct `sss_se05x_digest_t`**  
Message Digest operations

#### Public Members

*sss\_algorithm\_t* **algorithm**  
Algorithm to be applied, e.g. SHA1, SHA256

`SE05x_CryptoObjectID_t` **cryptoObjectId**  
Implementation specific part

`size_t` **digestFullLen**  
Full digest length per algorithm definition. This field is initialized along with algorithm.

*sss\_mode\_t* **mode**  
Mode of operation, e.g. Sign/Verify

*sss\_se05x\_session\_t* \***session**  
Virtual connection between application (user context) and specific security subsystem and function thereof.

## Struct `sss_se05x_key_store_t`

- Defined in file `_sss_inc_fsl_sss_se05x_types.h`

### Struct Documentation

#### **struct `sss_se05x_key_store_t`**

Store for secure and non secure key objects within a cryptographic system.

- A cryptographic system may have more than partitions to store such keys.

#### Public Members

##### **struct `_sss_se05x_object` \*`kekKey`**

In case the we are using Key Wrapping while injecting the keys, pointer to key used for wrapping

##### *sss\_se05x\_session\_t* \***session**

Pointer to the session

## Struct `sss_se05x_mac_t`

- Defined in file `_sss_inc_fsl_sss_se05x_types.h`

### Struct Documentation

#### **struct `sss_se05x_mac_t`**

Message Authentication Code.

#### Public Members

##### *sss\_algorithm\_t* **algorithm**

*copydoc sss\_mac\_t::algorithm*

##### SE05x\_CryptoObjectID\_t **cryptoObjectId**

Used crypto object ID for this operation

##### *sss\_se05x\_object\_t* \***keyObject**

*copydoc sss\_mac\_t::keyObject*

##### *sss\_mode\_t* **mode**

*copydoc sss\_mac\_t::mode*

##### *sss\_se05x\_session\_t* \***session**

*copydoc sss\_mac\_t::session*

## Struct `sss_se05x_object_t`

- Defined in `file_sss_inc_fsl_sss_se05x_types.h`

### Struct Documentation

#### **struct `sss_se05x_object_t`**

An object (secure / non-secure) within a Key Store.

#### **Public Members**

`uint32_t cipherType`

cipherType type from *`sss_cipher_type_t`*

`uint32_t curve_id`

If this is an ECC Key, the Curve ID of the key

`uint8_t isPersistent`

Whether this is a persistent or transient object

`uint32_t keyId`

Application specific key identifier. The keyId is kept in the key store along with the key data and other properties.

*`sss_se05x_key_store_t`* \***keyStore**

key store holding the data and other properties

`uint32_t objectType`

The type/part of object is referenced from *`sss_key_part_t`*

## Struct `sss_se05x_rng_context_t`

- Defined in `file_sss_inc_fsl_sss_se05x_types.h`

### Struct Documentation

#### **struct `sss_se05x_rng_context_t`**

Random number generator context

#### **Public Members**

*`sss_se05x_session_t`* \***session**

Pointer to the session

## Struct `sss_se05x_session_t`

- Defined in file `_sss_inc_fsl_sss_se05x_types.h`

### Struct Documentation

**struct `sss_se05x_session_t`**

Root session.

This is a *singleton* for each connection (physical/logical) to individual cryptographic system.

#### Public Members

`sss_se05x_tunnel_context_t` **\*ptun\_ctx**

In case connection is tunneled, context to the tunnel

`Se05xSession_t` **s\_ctx**

Connection context to SE050

`sss_type_t` **subsystem**

Indicates which security subsystem is selected to be used.

## Struct `sss_se05x_symmetric_t`

- Defined in file `_sss_inc_fsl_sss_se05x_types.h`

### Struct Documentation

**struct `sss_se05x_symmetric_t`**

Typedef for the symmetric crypto context.

#### Public Members

`sss_algorithm_t` **algorithm**

Algorithm to be applied, e.g AES\_ECB / CBC

`uint8_t` **cache\_data**[16]

Since underlying system only process in fixed chunks, cache them on host to complete the operation sanely

`size_t` **cache\_data\_len**

Length of bytes cached on host

`SE05x_CryptoObjectID_t` **cryptoObjectID**

Used crypto object ID for this operation

`sss_se05x_object_t` **\*keyObject**

Reference to key and it's properties.

`sss_mode_t` **mode**

Mode of operation, e.g Encryption/Decryption

`sss_se05x_session_t` **\*session**

Virtual connection between application (user context) and specific security subsystem and function thereof.

## Struct `sss_se05x_tunnel_context_t`

- Defined in file `_sss_inc_fsl_sss_se05x_types.h`

### Struct Documentation

**struct `sss_se05x_tunnel_context_t`**

Tunneling

Used for communication via another system.

#### Public Members

**struct `_sss_se05x_session *se05x_session`**

Pointer to the base SE050 SEssion

*sss\_tunnel\_dest\_t* **tunnelDest**

Where exactly this tunnel terminate to

## Struct `sss_session_t`

- Defined in file `_sss_inc_fsl_sss_api.h`

### Struct Documentation

**struct `sss_session_t`**

Root session.

This is a *singleton* for each connection (physical/logical) to individual cryptographic system.

#### Public Members

**uint8\_t `data`**[(0 + (1 \* sizeof(void \*)) + (1 \* sizeof(void \*)) + (8 \* sizeof(void \*)) + 16)]

**struct *sss\_session\_t*::[anonymous] extension**

Reserved memory for implementation specific extension

*sss\_type\_t* **subsystem**

Indicates which security subsystem is selected.

This is set when *sss\_session\_open* is successful



## Struct `sss_symmetric_t`

- Defined in `file_sss_inc_fsl_sss_api.h`

### Struct Documentation

#### **struct `sss_symmetric_t`**

Typedef for the symmetric crypto context.

#### **Public Members**

##### *sss\_algorithm\_t* **algorithm**

Algorithm to be applied, e.g AES\_ECB / CBC

`uint8_t data[(0 + (2 * sizeof(void *))) + (2 * sizeof(int)) + (2 * sizeof(void *)) + 16 + 4 + 16]`

##### **struct *sss\_symmetric\_t*::[anonymous] extension**

Reserved memory for implementation specific extension

##### *sss\_object\_t* \***keyObject**

Key to be used for the symmetric operation

##### *sss\_mode\_t* **mode**

Mode of operation, e.g Encryption/Decryption

##### *sss\_session\_t* \***session**

Virtual connection between application (user context) and specific security subsystem and function thereof.

## Struct `sss_tunnel_t`

- Defined in `file_sss_inc_fsl_sss_api.h`

### Struct Documentation

#### **struct `sss_tunnel_t`**

Tunneling

Used for communication via another system.

#### **Public Members**

`uint8_t data[(0 + (1 * sizeof(void *))) + (2 * sizeof(int)) + (2 * sizeof(void *)) + 16]`

##### **struct *sss\_tunnel\_t*::[anonymous] extension**

Reserved memory for implementation specific extension

##### *sss\_session\_t* \***session**

Pointer to the session

##### `uint32_t` **tunnelType**

TODO: More documentation

## Struct tlvHeader\_t

- Defined in file\_sss\_inc\_fsl\_sss\_api.h

## Struct Documentation

**struct tlvHeader\_t**  
Header for a IS716 APDU

### Public Members

uint8\_t **hdr**[0 + 1 + 1 + 1 + 1]  
ISO 7816 APDU Header

## Enums

### Enum SE05x\_AeadAlgo\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

## Enum Documentation

**enum SE05x\_AeadAlgo\_t**  
AEAD Algorithms  
*Values:*  
**kSE05x\_AeadAlgo\_NA** = 0  
Invalid  
**kSE05x\_AeadAlgo** = 0xB0

### Enum SE05x\_AppletConfig\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

## Enum Documentation

**enum SE05x\_AppletConfig\_t**  
Features which are available / enabled in the Applet  
*Values:*  
**kSE05x\_AppletConfig\_NA** = 0  
Invalid  
**kSE05x\_AppletConfig\_ECDA** = 0x0001  
Use of curve TPM\_ECC\_BN\_P256  
**kSE05x\_AppletConfig\_ECDSA\_ECDH\_ECDHE** = 0x0002  
EC DSA and DH support

**kSE05x\_AppletConfig\_EDDSA** = 0x0004  
 Use of curve RESERVED\_ID\_ECC\_ED\_25519

**kSE05x\_AppletConfig\_DH\_MONT** = 0x0008  
 Use of curve RESERVED\_ID\_ECC\_MONT\_DH\_25519

**kSE05x\_AppletConfig\_HMAC** = 0x0010  
 Writing HMACKey objects

**kSE05x\_AppletConfig\_RSA\_PLAIN** = 0x0020  
 Writing RSAKey objects

**kSE05x\_AppletConfig\_RSA\_CRT** = 0x0040  
 Writing RSAKey objects

**kSE05x\_AppletConfig\_AES** = 0x0080  
 Writing AESKey objects

**kSE05x\_AppletConfig\_DES** = 0x0100  
 Writing DESKey objects

**kSE05x\_AppletConfig\_PBKDF** = 0x0200  
 PBKDF2

**kSE05x\_AppletConfig\_TLS** = 0x0400  
 TLS Handshake support commands (see 4.16) in APDU Spec

**kSE05x\_AppletConfig\_MIFARE** = 0x0800  
 Mifare DESFire support (see 4.15) in APDU Spec

**kSE05x\_AppletConfig\_RFU1** = 0x1000  
 RFU1

**kSE05x\_AppletConfig\_I2CM** = 0x2000  
 I2C Master support (see 4.17) in APDU Spec

**kSE05x\_AppletConfig\_RFU2** = 0x4000  
 RFU2

## Enum SE05x\_AppletResID\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

## Enum Documentation

**enum SE05x\_AppletResID\_t**

Reserved identifiers of the Applet

*Values:*

**kSE05x\_AppletResID\_NA** = 0

Invalid

**kSE05x\_AppletResID\_TRANSPORT** = 0x7FFF0200

An authentication object which allows the user to switch LockState of the applet. The LockState defines whether the applet is transport locked or not.

**kSE05x\_AppletResID\_KP\_ECKEY\_USER** = 0x7FFF0201

A device unique NIST P-256 key pair which contains SK.SE.ECKA and PK.SE.ECKA in ECKEY session context.

**kSE05x\_AppletResID\_KP\_ECKEY\_IMPORT** = 0x7FFF0202

A device unique NIST P-256 key pair which contains SK.SE.ECKA and PK.SE.ECKA in ECKEY session context; A constant card challenge (all zeroes) is applicable.

**kSE05x\_AppletResID\_FEATURE** = 0x7FFF0204

An authentication object which allows the user to change the applet variant.

**kSE05x\_AppletResID\_FACTORY\_RESET** = 0x7FFF0205

An authentication object which allows the user to delete all objects, except trust provisioned by NXP objects.

**kSE05x\_AppletResID\_UNIQUE\_ID** = 0x7FFF0206

A BinaryFile Secure Object which holds the device unique ID. This file cannot be overwritten or deleted.

**kSE05x\_AppletResID\_PLATFORM\_SCP** = 0x7FFF0207

An authentication object which allows the user to change the platform SCP requirements, i.e. make platform SCP mandatory or not, using SetPlatformSCPRequest. Mandatory means full security, i.e. command & response MAC and encryption. Only SCP03 will be sufficient.

**kSE05x\_AppletResID\_I2CM\_ACCESS** = 0x7FFF0208

An authentication object which grants access to the I2C master feature. If the credential is not present, access to I2C master is allowed in general. Otherwise, a session using this credential shall be established and I2CM commands shall be sent within this session.

**kSE05x\_AppletResID\_RESTRICT** = 0x7FFF020A

An authentication object which grants access to the SetLockState command

## Enum SE05x\_AttestationAlgo\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

## Enum Documentation

**enum SE05x\_AttestationAlgo\_t**

Attestation

*Values:*

**kSE05x\_AttestationAlgo\_NA** = 0

**kSE05x\_AttestationAlgo\_EC\_PLAIN** = *kSE05x\_ECSignatureAlgo\_PLAIN*

**kSE05x\_AttestationAlgo\_EC\_SHA** = *kSE05x\_ECSignatureAlgo\_SHA*

**kSE05x\_AttestationAlgo\_EC\_SHA\_224** = *kSE05x\_ECSignatureAlgo\_SHA\_224*

**kSE05x\_AttestationAlgo\_EC\_SHA\_256** = *kSE05x\_ECSignatureAlgo\_SHA\_256*

**kSE05x\_AttestationAlgo\_EC\_SHA\_384** = *kSE05x\_ECSignatureAlgo\_SHA\_384*

**kSE05x\_AttestationAlgo\_EC\_SHA\_512** = *kSE05x\_ECSignatureAlgo\_SHA\_512*

**kSE05x\_AttestationAlgo\_ED25519PH\_SHA\_512** = *kSE05x\_EDSignatureAlgo\_ED25519PH\_SHA\_512*

**kSE05x\_AttestationAlgo\_ECDSA** = *kSE05x\_ECDAASignatureAlgo\_ECDSA*

**kSE05x\_AttestationAlgo\_RSA\_SHA1\_PKCS1\_PSS** = *kSE05x\_RSASignatureAlgo\_SHA1\_PKCS1\_PSS*

**kSE05x\_AttestationAlgo\_RSA\_SHA224\_PKCS1\_PSS** = *kSE05x\_RSASignatureAlgo\_SHA224\_PKCS1\_PSS*

**kSE05x\_AttestationAlgo\_RSA\_SHA256\_PKCS1\_PSS** = *kSE05x\_RSASignatureAlgo\_SHA256\_PKCS1\_PSS*

```

kSE05x_AttestationAlgo_RSA_SHA384_PKCS1_PSS = kSE05x_RSASignatureAlgo_SHA384_PKCS1_PSS
kSE05x_AttestationAlgo_RSA_SHA512_PKCS1_PSS = kSE05x_RSASignatureAlgo_SHA512_PKCS1_PSS
kSE05x_AttestationAlgo_RSA_SHA_224_PKCS1 = kSE05x_RSASignatureAlgo_SHA_224_PKCS1
kSE05x_AttestationAlgo_RSA_SHA_256_PKCS1 = kSE05x_RSASignatureAlgo_SHA_256_PKCS1
kSE05x_AttestationAlgo_RSA_SHA_384_PKCS1 = kSE05x_RSASignatureAlgo_SHA_384_PKCS1
kSE05x_AttestationAlgo_RSA_SHA_512_PKCS1 = kSE05x_RSASignatureAlgo_SHA_512_PKCS1

```

### Enum SE05x\_AttestationType\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

### Enum Documentation

**enum SE05x\_AttestationType\_t**

In case the read is attested

*Values:*

**kSE05x\_AttestationType\_None** = 0

**kSE05x\_AttestationType\_AUTH** = *kSE05x\_INS\_AUTH\_OBJECT*

### Enum SE05x\_Cipher\_Oper\_OneShot\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

### Enum Documentation

**enum SE05x\_Cipher\_Oper\_OneShot\_t**

One Shot operations helper

*Values:*

**kSE05x\_Cipher\_Oper\_OneShot\_NA** = 0

**kSE05x\_Cipher\_Oper\_OneShot\_Encrypt** = *kSE05x\_P2\_ENCRYPT\_ONESHOT*

**kSE05x\_Cipher\_Oper\_OneShot\_Decrypt** = *kSE05x\_P2\_DECRYPT\_ONESHOT*

### Enum SE05x\_Cipher\_Oper\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

## Enum Documentation

### enum SE05x\_Cipher\_Oper\_t

Cipher Operation.

Encrypt or decrypt

*Values:*

**kSE05x\_Cipher\_Oper\_NA** = 0

**kSE05x\_Cipher\_Oper\_Encrypt** = *kSE05x\_P2\_ENCRYPT*

**kSE05x\_Cipher\_Oper\_Decrypt** = *kSE05x\_P2\_DECRYPT*

### Enum SE05x\_CipherMode\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

## Enum Documentation

### enum SE05x\_CipherMode\_t

Symmetric cipher modes

*Values:*

**kSE05x\_CipherMode\_NA** = 0

Invalid

**kSE05x\_CipherMode\_DES\_CBC\_NOPAD** = 0x01

Typically using DESKey identifiers

**kSE05x\_CipherMode\_DES\_CBC\_ISO9797\_M1** = 0x02

Typically using DESKey identifiers

**kSE05x\_CipherMode\_DES\_CBC\_ISO9797\_M2** = 0x03

Typically using DESKey identifiers

**kSE05x\_CipherMode\_DES\_CBC\_PKCS5** = 0x04

NOT SUPPORTED

**kSE05x\_CipherMode\_DES\_ECB\_NOPAD** = 0x05

Typically using DESKey identifiers

**kSE05x\_CipherMode\_DES\_ECB\_ISO9797\_M1** = 0x06

NOT SUPPORTED

**kSE05x\_CipherMode\_DES\_ECB\_ISO9797\_M2** = 0x07

NOT SUPPORTED

**kSE05x\_CipherMode\_DES\_ECB\_PKCS5** = 0x08

NOT SUPPORTED

**kSE05x\_CipherMode\_AES\_ECB\_NOPAD** = 0x0E

Typically using AESKey identifiers

**kSE05x\_CipherMode\_AES\_CBC\_NOPAD** = 0x0D

Typically using AESKey identifiers

**kSE05x\_CipherMode\_AES\_CBC\_ISO9797\_M1** = 0x16

Typically using AESKey identifiers

**kSE05x\_CipherMode\_AES\_CBC\_ISO9797\_M2** = 0x17

Typically using AESKey identifiers

**kSE05x\_CipherMode\_AES\_CBC\_PKCS5** = 0x18

NOT SUPPORTED

**kSE05x\_CipherMode\_AES\_GCM** = 0xB0

Typically using AEAD mode

**kSE05x\_CipherMode\_AES\_CTR** = 0xF0

Typically using AESKey identifiers

## Enum SE05x\_CryptoContext\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

### Enum Documentation

**enum SE05x\_CryptoContext\_t**

Cryptographic context for operation

*Values:*

**kSE05x\_CryptoContext\_NA** = 0

Invalid

**kSE05x\_CryptoContext\_DIGEST** = 0x01

For DigestInit/DigestUpdate/DigestFinal

**kSE05x\_CryptoContext\_CIPHER** = 0x02

For CipherInit/CipherUpdate/CipherFinal

**kSE05x\_CryptoContext\_SIGNATURE** = 0x03

For MACInit/MACUpdate/MACFinal

**kSE05x\_CryptoContext\_AEAD** = 0x04

For AEADInit/AEADUpdate/AEADFinal

## Enum SE05x\_CryptoObject\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

### Enum Documentation

**enum SE05x\_CryptoObject\_t**

Crypto object identifiers

*Values:*

**kSE05x\_CryptoObject\_NA** = 0

Invalid

**kSE05x\_CryptoObject\_DIGEST\_SHA**

**kSE05x\_CryptoObject\_DIGEST\_SHA224**

**kSE05x\_CryptoObject\_DIGEST\_SHA256**

```

kSE05x_CryptoObject_DIGEST_SHA384
kSE05x_CryptoObject_DIGEST_SHA512
kSE05x_CryptoObject_DES_CBC_NOPAD
kSE05x_CryptoObject_DES_CBC_ISO9797_M1
kSE05x_CryptoObject_DES_CBC_ISO9797_M2
kSE05x_CryptoObject_DES_CBC_PKCS5
kSE05x_CryptoObject_DES_ECB_NOPAD
kSE05x_CryptoObject_DES_ECB_ISO9797_M1
kSE05x_CryptoObject_DES_ECB_ISO9797_M2
kSE05x_CryptoObject_DES_ECB_PKCS5
kSE05x_CryptoObject_AES_ECB_NOPAD
kSE05x_CryptoObject_AES_CBC_NOPAD
kSE05x_CryptoObject_AES_CBC_ISO9797_M1
kSE05x_CryptoObject_AES_CBC_ISO9797_M2
kSE05x_CryptoObject_AES_CBC_PKCS5
kSE05x_CryptoObject_AES_CTR
kSE05x_CryptoObject_HMAC_SHA1
kSE05x_CryptoObject_HMAC_SHA256
kSE05x_CryptoObject_HMAC_SHA384
kSE05x_CryptoObject_HMAC_SHA512
kSE05x_CryptoObject_CMAC_128
kSE05x_CryptoObject_AES_GCM

```

## Enum SE05x\_DigestMode\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

## Enum Documentation

**enum SE05x\_DigestMode\_t**

Hashing/Digest algorithms

*Values:*

**kSE05x\_DigestMode\_NA** = 0

Invalid

**kSE05x\_DigestMode\_NO\_HASH** = 0x00

**kSE05x\_DigestMode\_SHA** = 0x01

**kSE05x\_DigestMode\_SHA224** = 0x07

Not supported



`kSE05x_DigestMode_SHA256 = 0x04`

`kSE05x_DigestMode_SHA384 = 0x05`

`kSE05x_DigestMode_SHA512 = 0x06`

## Enum SE05x\_ECCurve\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

## Enum Documentation

**enum SE05x\_ECCurve\_t**

ECC Curve Identifiers

*Values:*

`kSE05x_ECCurve_NA = 0x00`

Invalid

`kSE05x_ECCurve_NIST_P192 = 0x01`

`kSE05x_ECCurve_NIST_P224 = 0x02`

`kSE05x_ECCurve_NIST_P256 = 0x03`

`kSE05x_ECCurve_NIST_P384 = 0x04`

`kSE05x_ECCurve_NIST_P521 = 0x05`

`kSE05x_ECCurve_Brainpool160 = 0x06`

`kSE05x_ECCurve_Brainpool192 = 0x07`

`kSE05x_ECCurve_Brainpool224 = 0x08`

`kSE05x_ECCurve_Brainpool256 = 0x09`

`kSE05x_ECCurve_Brainpool320 = 0x0A`

`kSE05x_ECCurve_Brainpool384 = 0x0B`

`kSE05x_ECCurve_Brainpool512 = 0x0C`

`kSE05x_ECCurve_Secp160k1 = 0x0D`

`kSE05x_ECCurve_Secp192k1 = 0x0E`

`kSE05x_ECCurve_Secp224k1 = 0x0F`

`kSE05x_ECCurve_Secp256k1 = 0x10`

`kSE05x_ECCurve_TPM_ECC_BN_P256 = 0x11`

`kSE05x_ECCurve_ECC_ED_25519 = 0x40`

Not Weierstrass

`kSE05x_ECCurve_ECC_MONT_DH_25519 = 0x41`

`kSE05x_ECCurve_ECC_MONT_DH_448 = 0x43`

Not Weierstrass

## Enum SE05x\_ECCurveParam\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

### Enum Documentation

**enum SE05x\_ECCurveParam\_t**

Parameters while setting the curve

*Values:*

**kSE05x\_ECCurveParam\_NA** = 0  
Invalid

**kSE05x\_ECCurveParam\_PARAM\_A** = 0x01

**kSE05x\_ECCurveParam\_PARAM\_B** = 0x02

**kSE05x\_ECCurveParam\_PARAM\_G** = 0x04

**kSE05x\_ECCurveParam\_PARAM\_N** = 0x08

**kSE05x\_ECCurveParam\_PARAM\_PRIME** = 0x10

## Enum SE05x\_ECDAASignatureAlgo\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

### Enum Documentation

**enum SE05x\_ECDAASignatureAlgo\_t**

Different signature algorithms for ECDAAs

*Values:*

**kSE05x\_ECDAASignatureAlgo\_NA** = 0  
Invalid

**kSE05x\_ECDAASignatureAlgo\_ECDAAs** = 0xF4  
Message input must be pre-hashed (using SHA256)

## Enum SE05x\_ECSignatureAlgo\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

## Enum Documentation

### enum SE05x\_ECSignatureAlgo\_t

Different signature algorithms for EC

*Values:*

**kSE05x\_ECSignatureAlgo\_NA** = 0  
Invalid

**kSE05x\_ECSignatureAlgo\_PLAIN** = 0x09  
NOT SUPPORTED

**kSE05x\_ECSignatureAlgo\_SHA** = 0x11

**kSE05x\_ECSignatureAlgo\_SHA\_224** = 0x25

**kSE05x\_ECSignatureAlgo\_SHA\_256** = 0x21

**kSE05x\_ECSignatureAlgo\_SHA\_384** = 0x22

**kSE05x\_ECSignatureAlgo\_SHA\_512** = 0x26

### Enum SE05x\_EDSignatureAlgo\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

## Enum Documentation

### enum SE05x\_EDSignatureAlgo\_t

Different signature algorithms for ED

*Values:*

**kSE05x\_EDSignatureAlgo\_NA** = 0  
Invalid

**kSE05x\_EDSignatureAlgo\_ED25519PH\_SHA\_512** = 0xA3  
Message input must be pre-hashed (using SHA512).

### Enum SE05x\_HealthCheckMode\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

## Enum Documentation

### enum SE05x\_HealthCheckMode\_t

Health check

*Values:*

**kSE05x\_HealthCheckMode\_NA** = 0  
Invalid

**kSE05x\_HealthCheckMode\_FIPS** = 0xF906  
Performs all on-demand self-tests. Can only be done when the module is in FIPS mode. When the test fails, the chip goes into TERMINATED state.

**kSE05x\_HealthCheckMode\_CODE\_SIGNATURE** = 0xFE01

Performs ROM integrity checks. When the test fails, the chip triggers the attack counter and the chip will reset.

**kSE05x\_HealthCheckMode\_DYNAMIC\_FLASH\_INTEGRITY** = 0xFD02

Performs flash integrity tests. When the test fails, the chip triggers the attack counter and the chip will reset.

**kSE05x\_HealthCheckMode\_SHIELDING** = 0xFB04

Performs tests on the active shield protection of the hardware. When the test fails, the chip triggers the attack counter and the chip will reset.

**kSE05x\_HealthCheckMode\_SENSOR** = 0xFA05

Performs self-tests on hardware sensors and reports the status.

**kSE05x\_HealthCheckMode\_SFR\_CHECK** = 0xFC03

Performs self-tests on the hardware registers. When the test fails, the chip triggers the attack counter and the chip will reset.

## Enum SE05x\_HkdfMode\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

## Enum Documentation

**enum SE05x\_HkdfMode\_t**

HKDF Mode

*Values:*

**kSE05x\_HkdfMode\_NA** = 0x00

Invalid

**kSE05x\_HkdfMode\_ExtractExpand** = 0x01

**kSE05x\_HkdfMode\_ExpandOnly** = 0x02

## Enum SE05x\_I2CM\_Baud\_Rate\_t

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_types.h

## Enum Documentation

**enum SE05x\_I2CM\_Baud\_Rate\_t**

Configuration for I2CM

*Values:*

**kSE05x\_I2CM\_Baud\_Rate\_100Khz** = 0

**kSE05x\_I2CM\_Baud\_Rate\_400Khz**

### Enum SE05x\_I2CM\_securityReq\_t

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_types.h

#### Enum Documentation

**enum SE05x\_I2CM\_securityReq\_t**

Additional operation on data read by I2C

*Values:*

**kSE05x\_Security\_None** = 0

**kSE05x\_Sign\_Request**

**kSE05x\_Sign\_Enc\_Request**

### Enum SE05x\_I2CM\_status\_t

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_types.h

#### Enum Documentation

**enum SE05x\_I2CM\_status\_t**

Status of I2CM Transaction

*Values:*

**kSE05x\_I2CM\_Success** = 0x5A

**kSE05x\_I2CM\_I2C\_Nack\_Fail** = 0x01

**kSE05x\_I2CM\_I2C\_Write\_Error** = 0x02

**kSE05x\_I2CM\_I2C\_Read\_Error** = 0x03

**kSE05x\_I2CM\_I2C\_Time\_Out\_Error** = 0x05

**kSE05x\_I2CM\_Invalid\_Tag** = 0x11

**kSE05x\_I2CM\_Invalid\_Length** = 0x12

**kSE05x\_I2CM\_Invalid\_Length\_Encode** = 0x13

**kSE05x\_I2CM\_I2C\_Config** = 0x21

### Enum SE05x\_I2CM\_TAG\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

## Enum Documentation

**enum SE05x\_I2CM\_TAG\_t**  
I2C Master micro operation.

*Values:*

**kSE05x\_TAG\_I2CM\_Config** = 0x01

**kSE05x\_TAG\_I2CM\_Write** = 0x03

**kSE05x\_TAG\_I2CM\_Read** = 0x04

## Enum SE05x\_I2CM\_TLV\_type\_t

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_types.h

## Enum Documentation

**enum SE05x\_I2CM\_TLV\_type\_t**  
Types of entries in an I2CM Transaction

*Values:*

**kSE05x\_I2CM\_None** = 0  
Do nothing

**kSE05x\_I2CM\_Configure**  
Configure the address, baudrate

**kSE05x\_I2CM\_Write** = 3  
Write to I2C Slave

**kSE05x\_I2CM\_Read**  
Read from I2C Slave

**kSE05x\_I2CM\_StructuralIssue** = 0xFF  
Response from SE05x that there is something wrong

## Enum SE05x\_INS\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

## Enum Documentation

**enum SE05x\_INS\_t**  
Values for INS in ISO7816 APDU

*Values:*

**kSE05x\_INS\_NA** = 0  
Invalid

**kSE05x\_INS\_MASK\_INS\_CHAR** = 0xE0  
3 MSBit for instruction characteristics.

**kSE05x\_INS\_MASK\_INSTRUCTION** = 0x1F

5 LsBit for instruction

**kSE05x\_INS\_TRANSIENT** = 0x80

Mask for transient object creation, can only be combined with INS\_WRITE.

**kSE05x\_INS\_AUTH\_OBJECT** = 0x40

Mask for authentication object creation, can only be combined with INS\_WRITE

**kSE05x\_INS\_ATTEST** = 0x20

Mask for getting attestation data.

**kSE05x\_INS\_WRITE** = 0x01

Write or create a persistent object.

**kSE05x\_INS\_READ** = 0x02

Read the object

**kSE05x\_INS\_CRYPT** = 0x03

Perform Security Operation

**kSE05x\_INS\_MGMT** = 0x04

General operation

**kSE05x\_INS\_PROCESS** = 0x05

Process session command

## Enum SE05x\_KeyPart\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

## Enum Documentation

**enum SE05x\_KeyPart\_t**

Part of the asymmetric key

*Values:*

**kSE05x\_KeyPart\_NA** = *kSE05x\_P1\_DEFAULT*

**kSE05x\_KeyPart\_Pair** = *kSE05x\_P1\_KEY\_PAIR*

Key pair (private key + public key)

**kSE05x\_KeyPart\_Private** = *kSE05x\_P1\_PRIVATE*

Private key

**kSE05x\_KeyPart\_Public** = *kSE05x\_P1\_PUBLIC*

Public key

## Enum SE05x\_LockIndicator\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

### Enum Documentation

**enum SE05x\_LockIndicator\_t**

Transient / Persistent lock

*Values:*

**kSE05x\_LockIndicator\_NA** = 0  
Invalid

**kSE05x\_LockIndicator\_TRANSIENT\_LOCK** = 0x01

**kSE05x\_LockIndicator\_PERSISTENT\_LOCK** = 0x02

## Enum SE05x\_LockState\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

### Enum Documentation

**enum SE05x\_LockState\_t**

Lock the sample (until unlocked )

*Values:*

**kSE05x\_LockState\_NA** = 0  
Invalid

**kSE05x\_LockState\_LOCKED** = 0x01

## Enum SE05x\_Mac\_Oper\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

### Enum Documentation

**enum SE05x\_Mac\_Oper\_t**

MAC operations

*Values:*

**kSE05x\_Mac\_Oper\_NA** = 0

**kSE05x\_Mac\_Oper\_Generate** = *kSE05x\_P2\_GENERATE*

**kSE05x\_Mac\_Oper\_Validate** = *kSE05x\_P2\_VALIDATE*



## Enum SE05x\_MACAlgo\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

### Enum Documentation

**enum SE05x\_MACAlgo\_t**

HMAC/CMAC Algorithms

*Values:*

**kSE05x\_MACAlgo\_NA** = 0

Invalid

**kSE05x\_MACAlgo\_HMAC\_SHA1** = 0x18

**kSE05x\_MACAlgo\_HMAC\_SHA256** = 0x19

**kSE05x\_MACAlgo\_HMAC\_SHA384** = 0x1A

**kSE05x\_MACAlgo\_HMAC\_SHA512** = 0x1B

**kSE05x\_MACAlgo\_CMAC\_128** = 0x31

## Enum SE05x\_MemoryType\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

### Enum Documentation

**enum SE05x\_MemoryType\_t**

Data for available memory

*Values:*

**kSE05x\_MemoryType\_NA** = 0

Invalid

**kSE05x\_MemoryType\_PERSISTENT** = 0x01

Persistent memory

**kSE05x\_MemoryType\_TRANSIENT\_RESET** = 0x02

Transient memory, clear on reset

**kSE05x\_MemoryType\_TRANSIENT\_DESELECT** = 0x03

Transient memory, clear on deselect

## Enum SE05x\_MemTyp\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

### Enum Documentation

#### enum SE05x\_MemTyp\_t

Type of memory. Used when we query available free size

*Values:*

**kSE05x\_MemTyp\_TRANSIENT\_RESET** = 0x01

Transient memory, clear on reset

**kSE05x\_MemTyp\_TRANSIENT\_DESELECT** = 0x02

Transient memory, clear on deselect

**kSE05x\_MemTyp\_PERSISTENT** = 0x03

Persistent memory

## Enum SE05x\_MoreIndicator\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

### Enum Documentation

#### enum SE05x\_MoreIndicator\_t

When there are more entries yet to be fetched from few of the APIs

*Values:*

**kSE05x\_MoreIndicator\_NA** = 0

Invalid

**kSE05x\_MoreIndicator\_NO\_MORE** = 0x01

No more data available

**kSE05x\_MoreIndicator\_MORE** = 0x02

More data available

## Enum SE05x\_Origin\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

## Enum Documentation

### enum SE05x\_Origin\_t

Where was this object originated

*Values:*

**kSE05x\_Origin\_NA** = 0  
Invalid

**kSE05x\_Origin\_EXTERNAL** = 0x01  
Generated outside the module.

**kSE05x\_Origin\_INTERNAL** = 0x02  
Generated inside the module.

**kSE05x\_Origin\_PROVISIONED** = 0x03  
Trust provisioned by NXP

### Enum SE05x\_P1\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

## Enum Documentation

### enum SE05x\_P1\_t

Values for P1 in ISO7816 APDU

*Values:*

**kSE05x\_P1\_NA** = 0  
Invalid

**kSE05x\_P1\_UNUSED** = 0x80  
Highest bit not used

**kSE05x\_P1\_MASK\_KEY\_TYPE** = 0x60  
2 MSBit for key type

**kSE05x\_P1\_MASK\_CRED\_TYPE** = 0x1F  
5 LSBit for credential type

**kSE05x\_P1\_KEY\_PAIR** = 0x60  
Key pair (private key + public key)

**kSE05x\_P1\_PRIVATE** = 0x40  
Private key

**kSE05x\_P1\_PUBLIC** = 0x20  
Public key

**kSE05x\_P1\_DEFAULT** = 0x00

**kSE05x\_P1\_EC** = 0x01

**kSE05x\_P1\_RSA** = 0x02

**kSE05x\_P1\_AES** = 0x03

**kSE05x\_P1\_DES** = 0x04

```

kSE05x_P1_HMAC = 0x05
kSE05x_P1_BINARY = 0x06
kSE05x_P1_UserID = 0x07
kSE05x_P1_COUNTER = 0x08
kSE05x_P1_PCR = 0x09
kSE05x_P1_CURVE = 0x0B
kSE05x_P1_SIGNATURE = 0x0C
kSE05x_P1_MAC = 0x0D
kSE05x_P1_CIPHER = 0x0E
kSE05x_P1_TLS = 0x0F
kSE05x_P1_CRYPTO_OBJ = 0x10
kSE05x_P1_AEAD = 0x11
 Applet >= 4.4
kSE05x_P1_AEAD_SP800_38D = 0x12
 Applet >= 4.4

```

## Enum SE05x\_P2\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

## Enum Documentation

### enum SE05x\_P2\_t

Values for P2 in ISO7816 APDU

*Values:*

```

kSE05x_P2_DEFAULT = 0x00
 Invalid
kSE05x_P2_GENERATE = 0x03
kSE05x_P2_CREATE = 0x04
kSE05x_P2_SIZE = 0x07
kSE05x_P2_SIGN = 0x09
kSE05x_P2_VERIFY = 0x0A
kSE05x_P2_INIT = 0x0B
kSE05x_P2_UPDATE = 0x0C
kSE05x_P2_FINAL = 0x0D
kSE05x_P2_ONESHOT = 0x0E
kSE05x_P2_DH = 0x0F
kSE05x_P2_DIVERSIFY = 0x10
kSE05x_P2_AUTH_FIRST_PART2 = 0x12

```

kSE05x\_P2\_AUTH\_NONFIRST\_PART2 = 0x13  
kSE05x\_P2\_DUMP\_KEY = 0x14  
kSE05x\_P2\_CHANGE\_KEY\_PART1 = 0x15  
kSE05x\_P2\_CHANGE\_KEY\_PART2 = 0x16  
kSE05x\_P2\_KILL\_AUTH = 0x17  
kSE05x\_P2\_IMPORT = 0x18  
kSE05x\_P2\_EXPORT = 0x19  
kSE05x\_P2\_SESSION\_CREATE = 0x1B  
kSE05x\_P2\_SESSION\_CLOSE = 0x1C  
kSE05x\_P2\_SESSION\_REFRESH = 0x1E  
kSE05x\_P2\_SESSION\_POLICY = 0x1F  
kSE05x\_P2\_VERSION = 0x20  
kSE05x\_P2\_MEMORY = 0x22  
kSE05x\_P2\_LIST = 0x25  
kSE05x\_P2\_TYPE = 0x26  
kSE05x\_P2\_EXIST = 0x27  
kSE05x\_P2\_DELETE\_OBJECT = 0x28  
kSE05x\_P2\_DELETE\_ALL = 0x2A  
kSE05x\_P2\_SESSION\_UserID = 0x2C  
kSE05x\_P2\_HKDF = 0x2D  
kSE05x\_P2\_PBKDF = 0x2E  
kSE05x\_P2\_HKDF\_EXPAND\_ONLY = 0x2F  
kSE05x\_P2\_I2CM = 0x30  
kSE05x\_P2\_I2CM\_ATTESTED = 0x31  
kSE05x\_P2\_MAC = 0x32  
kSE05x\_P2\_UNLOCK\_CHALLENGE = 0x33  
kSE05x\_P2\_CURVE\_LIST = 0x34  
kSE05x\_P2\_SIGN\_ECDA = 0x35  
kSE05x\_P2\_ID = 0x36  
kSE05x\_P2\_ENCRYPT\_ONESHOT = 0x37  
kSE05x\_P2\_DECRYPT\_ONESHOT = 0x38  
kSE05x\_P2\_ATTEST = 0x3A  
kSE05x\_P2\_ATTRIBUTES = 0x3B  
kSE05x\_P2\_CPLC = 0x3C  
kSE05x\_P2\_TIME = 0x3D  
kSE05x\_P2\_TRANSPORT = 0x3E

```

kSE05x_P2_VARIANT = 0x3F
kSE05x_P2_PARAM = 0x40
kSE05x_P2_DELETE_CURVE = 0x41
kSE05x_P2_ENCRYPT = 0x42
kSE05x_P2_DECRYPT = 0x43
kSE05x_P2_VALIDATE = 0x44
kSE05x_P2_GENERATE_ONESHOT = 0x45
kSE05x_P2_VALIDATE_ONESHOT = 0x46
kSE05x_P2_CRYPTO_LIST = 0x47
kSE05x_P2_RANDOM = 0x49
kSE05x_P2_TLS_PMS = 0x4A
kSE05x_P2_TLS_PRF_CLI_HELLO = 0x4B
kSE05x_P2_TLS_PRF_SRV_HELLO = 0x4C
kSE05x_P2_TLS_PRF_CLI_RND = 0x4D
kSE05x_P2_TLS_PRF_SRV_RND = 0x4E
kSE05x_P2_RAW = 0x4F
kSE05x_P2_IMPORT_EXT = 0x51
kSE05x_P2_SCP = 0x52
kSE05x_P2_AUTH_FIRST_PART1 = 0x53
kSE05x_P2_AUTH_NONFIRST_PART1 = 0x54
kSE05x_P2_CM_COMMAND = 0x55
kSE05x_P2_MODE_OF_OPERATION = 0x56
kSE05x_P2_RESTRICT = 0x57
kSE05x_P2_SANITY = 0x58
kSE05x_P2_DH_REVERSE = 0x59

```

## Enum SE05x\_PlatformSCPRequest\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

## Enum Documentation

**enum SE05x\_PlatformSCPRequest\_t**

Mandate platform SCP or not

*Values:*

```

kSE05x_PlatformSCPRequest_NA = 0
 Invalid

```

**kSE05x\_PlatformSCPRequest\_REQUIRED** = 0x01

Platform SCP is required (full enc & MAC)

**kSE05x\_PlatformSCPRequest\_NOT\_REQUIRED** = 0x02

No platform SCP required.

### Enum SE05x\_RestrictMode\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

### Enum Documentation

**enum SE05x\_RestrictMode\_t**

Applet >= 4.4

See Se05x\_API\_DisableObjCreation

*Values:*

**kSE05x\_RestrictMode\_NA** = 0

**kSE05x\_RestrictMode\_RESTRICT\_NEW** = 0x01

**kSE05x\_RestrictMode\_RESTRICT\_ALL** = 0x02

### Enum SE05x\_Result\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

### Enum Documentation

**enum SE05x\_Result\_t**

Result of operations

*Values:*

**kSE05x\_Result\_NA** = 0

Invalid

**kSE05x\_Result\_SUCCESS** = 0x01

**kSE05x\_Result\_FAILURE** = 0x02

### Enum SE05x\_RSABitLength\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

## Enum Documentation

### enum SE05x\_RSABitLength\_t

Size of RSA Key Objects

*Values:*

**kSE05x\_RSABitLength\_NA** = 0

Invalid

**kSE05x\_RSABitLength\_512** = 512

**kSE05x\_RSABitLength\_1024** = 1024

**kSE05x\_RSABitLength\_1152** = 1152

**kSE05x\_RSABitLength\_2048** = 2048

**kSE05x\_RSABitLength\_3072** = 3072

**kSE05x\_RSABitLength\_4096** = 4096

### Enum SE05x\_RSAEncryptionAlgo\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

## Enum Documentation

### enum SE05x\_RSAEncryptionAlgo\_t

Different encryption/decryption algorithms for RSA

*Values:*

**kSE05x\_RSAEncryptionAlgo\_NA** = 0

Invalid

**kSE05x\_RSAEncryptionAlgo\_NO\_PAD** = 0x0C

Plain RSA, padding required on host.

**kSE05x\_RSAEncryptionAlgo\_PKCS1** = 0x0A

RFC8017: RSAES-PKCS1-v1\_5

**kSE05x\_RSAEncryptionAlgo\_PKCS1\_OAEP** = 0x0F

RFC8017: RSAES-OAEP

### Enum SE05x\_RSAKeyComponent\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h



## Enum Documentation

**enum SE05x\_RSASKeyComponent\_t**

Part of the RSA Key Objects

*Values:*

**kSE05x\_RSASKeyComponent\_NA** = 0xFF

Invalid

**kSE05x\_RSASKeyComponent\_MOD** = 0x00

Modulus

**kSE05x\_RSASKeyComponent\_PUB\_EXP** = 0x01

Public key exponent

**kSE05x\_RSASKeyComponent\_PRIV\_EXP** = 0x02

Private key exponent

**kSE05x\_RSASKeyComponent\_P** = 0x03

CRT component p

**kSE05x\_RSASKeyComponent\_Q** = 0x04

CRT component q

**kSE05x\_RSASKeyComponent\_DP** = 0x05

CRT component dp

**kSE05x\_RSASKeyComponent\_DQ** = 0x06

CRT component dq

**kSE05x\_RSASKeyComponent\_INVQ** = 0x07

CRT component q\_inv

## Enum SE05x\_RSASKeyFormat\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

## Enum Documentation

**enum SE05x\_RSASKeyFormat\_t**

RSA Key format

*Values:*

**kSE05x\_RSASKeyFormat\_CRT** = *kSE05x\_P2\_DEFAULT*

**kSE05x\_RSASKeyFormat\_RAW** = *kSE05x\_P2\_RAW*

## Enum SE05x\_RSAPubKeyComp\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

### Enum Documentation

**enum SE05x\_RSAPubKeyComp\_t**

Plain RSA, padding required on host. RFC8017: RSAES-OAEP Public part of RSA Keys

*Values:*

**kSE05x\_RSAPubKeyComp\_NA** = 0

**kSE05x\_RSAPubKeyComp\_MOD** = *kSE05x\_RSAKeyComponent\_MOD*

**kSE05x\_RSAPubKeyComp\_PUB\_EXP** = *kSE05x\_RSAKeyComponent\_PUB\_EXP*

## Enum SE05x\_RSASignAlgo\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

### Enum Documentation

**enum SE05x\_RSASignAlgo\_t**

Algorithms for RSA Signature

*Values:*

**kSE05x\_RSASignAlgo\_NA** = 0

Invalid

**kSE05x\_RSASignAlgo\_SHA1\_PKCS1\_PSS** = 0x15

RFC8017: RSASSA-PSS

**kSE05x\_RSASignAlgo\_SHA224\_PKCS1\_PSS** = 0x2B

RFC8017: RSASSA-PSS

**kSE05x\_RSASignAlgo\_SHA256\_PKCS1\_PSS** = 0x2C

RFC8017: RSASSA-PSS

**kSE05x\_RSASignAlgo\_SHA384\_PKCS1\_PSS** = 0x2D

RFC8017: RSASSA-PSS

**kSE05x\_RSASignAlgo\_SHA512\_PKCS1\_PSS** = 0x2E

RFC8017: RSASSA-PSS

**kSE05x\_RSASignAlgo\_SHA\_224\_PKCS1** = 0x27

RFC8017: RSASSA-PKCS1-v1\_5

**kSE05x\_RSASignAlgo\_SHA\_256\_PKCS1** = 0x28

RFC8017: RSASSA-PKCS1-v1\_5

**kSE05x\_RSASignAlgo\_SHA\_384\_PKCS1** = 0x29

RFC8017: RSASSA-PKCS1-v1\_5

**kSE05x\_RSASignAlgo\_SHA\_512\_PKCS1** = 0x2A

RFC8017: RSASSA-PKCS1-v1\_5

## Enum SE05x\_RSASignatureAlgo\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

## Enum Documentation

### enum SE05x\_RSASignatureAlgo\_t

Different signature algorithms for RSA

*Values:*

**kSE05x\_RSASignatureAlgo\_NA** = 0  
Invalid

**kSE05x\_RSASignatureAlgo\_SHA1\_PKCS1\_PSS** = 0x15  
RFC8017: RSASSA-PSS

**kSE05x\_RSASignatureAlgo\_SHA224\_PKCS1\_PSS** = 0x2B  
RFC8017: RSASSA-PSS

**kSE05x\_RSASignatureAlgo\_SHA256\_PKCS1\_PSS** = 0x2C  
RFC8017: RSASSA-PSS

**kSE05x\_RSASignatureAlgo\_SHA384\_PKCS1\_PSS** = 0x2D  
RFC8017: RSASSA-PSS

**kSE05x\_RSASignatureAlgo\_SHA512\_PKCS1\_PSS** = 0x2E  
RFC8017: RSASSA-PSS

**kSE05x\_RSASignatureAlgo\_SHA1\_PKCS1** = 0x0A  
RFC8017: RSASSA-PKCS1-v1\_5

**kSE05x\_RSASignatureAlgo\_SHA\_224\_PKCS1** = 0x27  
RFC8017: RSASSA-PKCS1-v1\_5

**kSE05x\_RSASignatureAlgo\_SHA\_256\_PKCS1** = 0x28  
RFC8017: RSASSA-PKCS1-v1\_5

**kSE05x\_RSASignatureAlgo\_SHA\_384\_PKCS1** = 0x29  
RFC8017: RSASSA-PKCS1-v1\_5

**kSE05x\_RSASignatureAlgo\_SHA\_512\_PKCS1** = 0x2A  
RFC8017: RSASSA-PKCS1-v1\_5

## Enum SE05x\_SecObjTyp\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

## Enum Documentation

**enum SE05x\_SecObjTyp\_t**

Type of Object

*Values:*

```
kSE05x_SecObjTyp_EC_KEY_PAIR = 0x01
kSE05x_SecObjTyp_EC_PRIV_KEY = 0x02
kSE05x_SecObjTyp_EC_PUB_KEY = 0x03
kSE05x_SecObjTyp_RSA_KEY_PAIR = 0x04
kSE05x_SecObjTyp_RSA_KEY_PAIR_CRT = 0x05
kSE05x_SecObjTyp_RSA_PRIV_KEY = 0x06
kSE05x_SecObjTyp_RSA_PRIV_KEY_CRT = 0x07
kSE05x_SecObjTyp_RSA_PUB_KEY = 0x08
kSE05x_SecObjTyp_AES_KEY = 0x09
kSE05x_SecObjTyp_DES_KEY = 0x0A
kSE05x_SecObjTyp_BINARY_FILE = 0x0B
kSE05x_SecObjTyp_UserID = 0x0C
kSE05x_SecObjTyp_COUNTER = 0x0D
kSE05x_SecObjTyp_PCR = 0x0F
kSE05x_SecObjTyp_CURVE = 0x10
kSE05x_SecObjTyp_HMAC_KEY = 0x11
```

**Enum SE05x\_SetIndicator\_t**

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

## Enum Documentation

**enum SE05x\_SetIndicator\_t**

TODO

*Values:*

```
kSE05x_SetIndicator_NA = 0
 Invalid
kSE05x_SetIndicator_NOT_SET = 0x01
kSE05x_SetIndicator_SET = 0x02
```

## Enum SE05x\_SW12\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

### Enum Documentation

#### enum SE05x\_SW12\_t

Mapping of 2 byte return code

*Values:*

**kSE05x\_SW12\_NA** = 0

Invalid

**kSE05x\_SW12\_NO\_ERROR** = 0x9000

No Error

**kSE05x\_SW12\_CONDITIONS\_NOT\_SATISFIED** = 0x6985

Conditions not satisfied

**kSE05x\_SW12\_SECURITY\_STATUS** = 0x6982

Security status not satisfied.

**kSE05x\_SW12\_WRONG\_DATA** = 0x6A80

Wrong data provided.

**kSE05x\_SW12\_DATA\_INVALID** = 0x6984

Data invalid - policy set invalid for the given object

**kSE05x\_SW12\_COMMAND\_NOT\_ALLOWED** = 0x6986

Command not allowed - access denied based on object policy

## Enum SE05x\_SymmKeyType\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

### Enum Documentation

#### enum SE05x\_SymmKeyType\_t

Symmetric keys

*Values:*

**kSE05x\_SymmKeyType\_AES** = *kSE05x\_P1\_AES*

**kSE05x\_SymmKeyType\_DES** = *kSE05x\_P1\_DES*

**kSE05x\_SymmKeyType\_HMAC** = *kSE05x\_P1\_HMAC*

**kSE05x\_SymmKeyType\_CMAC** = *kSE05x\_P1\_AES*

## Enum SE05x\_TAG\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

## Enum Documentation

### enum SE05x\_TAG\_t

Different TAG Values to talk to SE05X IoT Applet

*Values:*

**kSE05x\_TAG\_NA** = 0

Invalid

**kSE05x\_TAG\_SESSION\_ID** = 0x10

**kSE05x\_TAG\_POLICY** = 0x11

**kSE05x\_TAG\_MAX\_ATTEMPTS** = 0x12

**kSE05x\_TAG\_IMPORT\_AUTH\_DATA** = 0x13

**kSE05x\_TAG\_IMPORT\_AUTH\_KEY\_ID** = 0x14

**kSE05x\_TAG\_1** = 0x41

**kSE05x\_TAG\_2** = 0x42

**kSE05x\_TAG\_3** = 0x43

**kSE05x\_TAG\_4** = 0x44

**kSE05x\_TAG\_5** = 0x45

**kSE05x\_TAG\_6** = 0x46

**kSE05x\_TAG\_7** = 0x47

**kSE05x\_TAG\_8** = 0x48

**kSE05x\_TAG\_9** = 0x49

**kSE05x\_TAG\_10** = 0x4A

**kSE05x\_GP\_TAG\_CONTRL\_REF\_PARM** = 0xA6

**kSE05x\_GP\_TAG\_AID** = 0x4F

**kSE05x\_GP\_TAG\_KEY\_TYPE** = 0x80

**kSE05x\_GP\_TAG\_KEY\_LEN** = 0x81

**kSE05x\_GP\_TAG\_GET\_DATA** = 0x83

**kSE05x\_GP\_TAG\_DR\_SE** = 0x85

**kSE05x\_GP\_TAG\_RECEIPT** = 0x86

**kSE05x\_GP\_TAG\_SCP\_PARAMS** = 0x90

## Enum SE05x\_TLSPerformPRFType\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

### Enum Documentation

**enum SE05x\_TLSPerformPRFType\_t**

TLS Perform PRF

*Values:*

**kSE05x\_TLS\_PRF\_NA** = 0

**kSE05x\_TLS\_PRF\_CLI\_HELLO** = *kSE05x\_P2\_TLS\_PRF\_CLI\_HELLO*

**kSE05x\_TLS\_PRF\_SRV\_HELLO** = *kSE05x\_P2\_TLS\_PRF\_SRV\_HELLO*

**kSE05x\_TLS\_PRF\_CLI\_RND** = *kSE05x\_P2\_TLS\_PRF\_CLI\_RND*

**kSE05x\_TLS\_PRF\_SRV\_RND** = *kSE05x\_P2\_TLS\_PRF\_SRV\_RND*

## Enum SE05x\_TransientIndicator\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

### Enum Documentation

**enum SE05x\_TransientIndicator\_t**

Whether object is transient or persistent

*Values:*

**kSE05x\_TransientIndicator\_NA** = 0

Invalid

**kSE05x\_TransientIndicator\_PERSISTENT** = 0x01

**kSE05x\_TransientIndicator\_TRANSIENT** = 0x02

## Enum SE05x\_TransientType\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

### Enum Documentation

**enum SE05x\_TransientType\_t**

Whether key is transient or persistent

*Values:*

**kSE05x\_TransientType\_Persistent** = 0

**kSE05x\_TransientType\_Transient** = *kSE05x\_INS\_TRANSIENT*

## Enum SE\_AuthType\_t

- Defined in file\_hostlib\_hostLib\_inc\_nxScp03\_Types.h

## Enum Documentation

### enum SE\_AuthType\_t

*Values:*

**kSSS\_AuthType\_None** = 0

**kSSS\_AuthType\_SCP03** = 1  
Global platform SCP03

**kSSS\_AuthType\_ID** = 2  
(e.g. SE05X) UserID based connection

**kSSS\_AuthType\_AESKey** = 3  
(e.g. SE05X) Use AESKey for user authentication

Earlier this was called kSSS\_AuthType\_AppletSCP03

**kSSS\_AuthType\_ECKey** = 4  
(e.g. SE05X) Use ECKey for user authentication

Earlier this was called kSSS\_AuthType\_FastSCP

**kSSS\_AuthType\_INT\_ECKey\_Counter** = 0x14  
Used internally, not to be set/used by user.

For the versions of the applet where we have to add the a counter during KDF.

**kSSS\_SIZE** = 0x7FFFFFFF

## Enum sss\_access\_permission\_t

- Defined in file\_sss\_inc\_fsl\_sss\_api.h

## Enum Documentation

### enum sss\_access\_permission\_t

Permissions of an object

*Values:*

**kAccessPermission\_SSS\_Read** = (1u << 0)  
Can read (applicable) contents of the key.

@note This is not same as @ref kAccessPermission\_SSS\_Use.

Without reading, the object, the key can be used.

**kAccessPermission\_SSS\_Write** = (1u << 1)  
Can change the value of an object



```

kAccessPermission_SSS_Use = (1u << 2)
 Can use an object

kAccessPermission_SSS_Delete = (1u << 3)
 Can delete an object

kAccessPermission_SSS_ChangeAttributes = (1u << 4)
 Can change permissions applicable to an object

```

## Enum sss\_algorithm\_t

- Defined in file\_sss\_inc\_fsl\_sss\_api.h

## Enum Documentation

**enum sss\_algorithm\_t**

Cryptographic algorithm to be applied

*Values:*

```

kAlgorithm_None

kAlgorithm_SSS_AES_ECB = SSS_ENUM_ALGORITHM(AES, 0x01)
kAlgorithm_SSS_AES_CBC = SSS_ENUM_ALGORITHM(AES, 0x02)
kAlgorithm_SSS_AES_CTR = SSS_ENUM_ALGORITHM(AES, 0x03)
kAlgorithm_SSS_AES_GCM = SSS_ENUM_ALGORITHM(AES, 0x04)
kAlgorithm_SSS_AES_CCM = SSS_ENUM_ALGORITHM(AES, 0x05)
kAlgorithm_SSS_CHACHA_POLY = SSS_ENUM_ALGORITHM(CHACHA, 0x01)
kAlgorithm_SSS_DES_ECB = SSS_ENUM_ALGORITHM(DES, 0x01)
kAlgorithm_SSS_DES_CBC = SSS_ENUM_ALGORITHM(DES, 0x02)
kAlgorithm_SSS_DES3_ECB = SSS_ENUM_ALGORITHM(DES, 0x03)
kAlgorithm_SSS_DES3_CBC = SSS_ENUM_ALGORITHM(DES, 0x04)
kAlgorithm_SSS_SHA1 = SSS_ENUM_ALGORITHM(SHA, 0x01)
kAlgorithm_SSS_SHA224 = SSS_ENUM_ALGORITHM(SHA, 0x02)
kAlgorithm_SSS_SHA256 = SSS_ENUM_ALGORITHM(SHA, 0x03)
kAlgorithm_SSS_SHA384 = SSS_ENUM_ALGORITHM(SHA, 0x04)
kAlgorithm_SSS_SHA512 = SSS_ENUM_ALGORITHM(SHA, 0x05)
kAlgorithm_SSS_CMAC_AES = SSS_ENUM_ALGORITHM(MAC, 0x01)
kAlgorithm_SSS_HMAC_SHA1 = SSS_ENUM_ALGORITHM(MAC, 0x02)
kAlgorithm_SSS_HMAC_SHA224 = SSS_ENUM_ALGORITHM(MAC, 0x03)
kAlgorithm_SSS_HMAC_SHA256 = SSS_ENUM_ALGORITHM(MAC, 0x04)
kAlgorithm_SSS_HMAC_SHA384 = SSS_ENUM_ALGORITHM(MAC, 0x05)
kAlgorithm_SSS_HMAC_SHA512 = SSS_ENUM_ALGORITHM(MAC, 0x06)
kAlgorithm_SSS_DH = SSS_ENUM_ALGORITHM(DH, 0x01)

```

```

kAlgorithm_SSS_ECDH = SSS_ENUM_ALGORITHM(DH, 0x02)
kAlgorithm_SSS_DSA_SHA1 = SSS_ENUM_ALGORITHM(DSA, 0x01)
kAlgorithm_SSS_DSA_SHA224 = SSS_ENUM_ALGORITHM(DSA, 0x02)
kAlgorithm_SSS_DSA_SHA256 = SSS_ENUM_ALGORITHM(DSA, 0x03)
kAlgorithm_SSS_RSASSA_PKCS1_V1_5_NO_HASH = SSS_ENUM_ALGORITHM(RSASSA_PKCS1_V1_5, 0x01)
kAlgorithm_SSS_RSASSA_PKCS1_V1_5_SHA1 = SSS_ENUM_ALGORITHM(RSASSA_PKCS1_V1_5, 0x02)
kAlgorithm_SSS_RSASSA_PKCS1_V1_5_SHA224 = SSS_ENUM_ALGORITHM(RSASSA_PKCS1_V1_5, 0x03)
kAlgorithm_SSS_RSASSA_PKCS1_V1_5_SHA256 = SSS_ENUM_ALGORITHM(RSASSA_PKCS1_V1_5, 0x04)
kAlgorithm_SSS_RSASSA_PKCS1_V1_5_SHA384 = SSS_ENUM_ALGORITHM(RSASSA_PKCS1_V1_5, 0x05)
kAlgorithm_SSS_RSASSA_PKCS1_V1_5_SHA512 = SSS_ENUM_ALGORITHM(RSASSA_PKCS1_V1_5, 0x06)
kAlgorithm_SSS_RSASSA_PKCS1_PSS_MGF1_SHA1 = SSS_ENUM_ALGORITHM(RSASSA_PKCS1_PSS_MGF1, 0x01)
kAlgorithm_SSS_RSASSA_PKCS1_PSS_MGF1_SHA224 = SSS_ENUM_ALGORITHM(RSASSA_PKCS1_PSS_MGF1, 0x02)
kAlgorithm_SSS_RSASSA_PKCS1_PSS_MGF1_SHA256 = SSS_ENUM_ALGORITHM(RSASSA_PKCS1_PSS_MGF1, 0x03)
kAlgorithm_SSS_RSASSA_PKCS1_PSS_MGF1_SHA384 = SSS_ENUM_ALGORITHM(RSASSA_PKCS1_PSS_MGF1, 0x04)
kAlgorithm_SSS_RSASSA_PKCS1_PSS_MGF1_SHA512 = SSS_ENUM_ALGORITHM(RSASSA_PKCS1_PSS_MGF1, 0x05)
kAlgorithm_SSS_RSAES_PKCS1_OAEP_SHA1 = SSS_ENUM_ALGORITHM(RSAES_PKCS1_OAEP, 0x01)
kAlgorithm_SSS_RSAES_PKCS1_OAEP_SHA224 = SSS_ENUM_ALGORITHM(RSAES_PKCS1_OAEP, 0x02)
kAlgorithm_SSS_RSAES_PKCS1_OAEP_SHA256 = SSS_ENUM_ALGORITHM(RSAES_PKCS1_OAEP, 0x03)
kAlgorithm_SSS_RSAES_PKCS1_OAEP_SHA384 = SSS_ENUM_ALGORITHM(RSAES_PKCS1_OAEP, 0x04)
kAlgorithm_SSS_RSAES_PKCS1_OAEP_SHA512 = SSS_ENUM_ALGORITHM(RSAES_PKCS1_OAEP, 0x05)
kAlgorithm_SSS_RSAES_PKCS1_V1_5_SHA1 = SSS_ENUM_ALGORITHM(RSAES_PKCS1_V1_5, 0x01)
kAlgorithm_SSS_RSAES_PKCS1_V1_5_SHA224 = SSS_ENUM_ALGORITHM(RSAES_PKCS1_V1_5, 0x02)
kAlgorithm_SSS_RSAES_PKCS1_V1_5_SHA256 = SSS_ENUM_ALGORITHM(RSAES_PKCS1_V1_5, 0x03)
kAlgorithm_SSS_RSAES_PKCS1_V1_5_SHA384 = SSS_ENUM_ALGORITHM(RSAES_PKCS1_V1_5, 0x04)
kAlgorithm_SSS_RSAES_PKCS1_V1_5_SHA512 = SSS_ENUM_ALGORITHM(RSAES_PKCS1_V1_5, 0x05)
kAlgorithm_SSS_RSASSA_NO_PADDING = SSS_ENUM_ALGORITHM(RSASSA_NO_PADDING, 0x01)
kAlgorithm_SSS_ECDSA_SHA1 = SSS_ENUM_ALGORITHM(ECDSA, 0x01)
kAlgorithm_SSS_ECDSA_SHA224 = SSS_ENUM_ALGORITHM(ECDSA, 0x02)
kAlgorithm_SSS_ECDSA_SHA256 = SSS_ENUM_ALGORITHM(ECDSA, 0x03)
kAlgorithm_SSS_ECDSA_SHA384 = SSS_ENUM_ALGORITHM(ECDSA, 0x04)
kAlgorithm_SSS_ECDSA_SHA512 = SSS_ENUM_ALGORITHM(ECDSA, 0x05)

```

## Enum sss\_cipher\_type\_t

- Defined in file\_sss\_inc\_fsl\_sss\_api.h

## Enum Documentation

### enum sss\_cipher\_type\_t

For all cipher types, key bit length is provides at the time key is inserted/generated

*Values:*

**kSSS\_CipherType\_NONE**

**kSSS\_CipherType\_AES** = 10

**kSSS\_CipherType\_DES** = 12

**kSSS\_CipherType\_CMAC** = 20

**kSSS\_CipherType\_HMAC** = 21

**kSSS\_CipherType\_MAC** = 30

**kSSS\_CipherType\_RSA** = 31

**kSSS\_CipherType\_RSA\_CRT** = 32

RSA RAW format

**kSSS\_CipherType\_EC\_NIST\_P** = 40

RSA CRT format

**kSSS\_CipherType\_EC\_NIST\_K** = 41

Keys Part of NIST-P Family

**kSSS\_CipherType\_EC\_MONTGOMERY** = 50

Keys Part of NIST-K Family Montgomery Key,

**kSSS\_CipherType\_EC\_TWISTED\_ED** = 51

twisted Edwards form elliptic curve public key

**kSSS\_CipherType\_EC\_BRAINPOOL** = 52

Brainpool form elliptic curve public key

**kSSS\_CipherType\_EC\_BARRETO\_NAEHRIG** = 53

Barreto Naehrig curve

**kSSS\_CipherType\_UserID** = 70

**kSSS\_CipherType\_Certificate** = 71

**kSSS\_CipherType\_Binary** = 72

**kSSS\_CipherType\_Count** = 73

**kSSS\_CipherType\_PCR** = 74

**kSSS\_CipherType\_ReservedPin** = 75

## Enum sss\_connection\_type\_t

- Defined in file\_sss\_inc\_fsl\_sss\_api.h

### Enum Documentation

**enum sss\_connection\_type\_t**

Destintion connection type

*Values:*

**kSSS\_ConnectionType\_Plain**

**kSSS\_ConnectionType\_Password**

**kSSS\_ConnectionType\_Encrypted**

## Enum sss\_key\_object\_mode\_t

- Defined in file\_sss\_inc\_fsl\_sss\_api.h

### Enum Documentation

**enum sss\_key\_object\_mode\_t**

Persistent / Non persistent mode of a key

*Values:*

**kKeyObject\_Mode\_None = 0**

kKeyObject\_Mode\_None

**kKeyObject\_Mode\_Persistent = 1**

Key object will be persisted in memory and will retain it's value after a closed session

**kKeyObject\_Mode\_Transient = 2**

Key Object will be stored in RAM. It will lose it's contents after a session is closed

## Enum sss\_key\_part\_t

- Defined in file\_sss\_inc\_fsl\_sss\_api.h

### Enum Documentation

**enum sss\_key\_part\_t**

Part of a key

*Values:*

**kSSS\_KeyPart\_NONE**

**kSSS\_KeyPart\_Default = 1**

Applicable where we have UserID, PIN, Binary Files, Certificates, Symmetric Keys, PCR

**kSSS\_KeyPart\_Public = 2**

Public part of asymmetric key

**kSSS\_KeyPart\_Private** = 3  
Private only part of asymmetric key

**kSSS\_KeyPart\_Pair** = 4  
Both, public and private part of asymmetric key

### Enum sss\_key\_store\_prop\_au8\_t

- Defined in file\_sss\_inc\_fsl\_sss\_api.h

### Enum Documentation

**enum sss\_key\_store\_prop\_au8\_t**  
properties of a Key Store that return array

*Values:*

**kSSS\_KeyStoreProp\_au8\_Optional\_Start** = 0x00FFFFFFu  
Optional Properties Start

### Enum sss\_mode\_t

- Defined in file\_sss\_inc\_fsl\_sss\_api.h

### Enum Documentation

**enum sss\_mode\_t**  
High level algorithmic operations.

Augmented by *sss\_algorithm\_t*

*Values:*

**kMode\_SSS\_Encrypt** = 1  
Encrypt.

**kMode\_SSS\_Decrypt** = 2  
Decrypt.

**kMode\_SSS\_Sign** = 3  
Sign.

**kMode\_SSS\_Verify** = 4  
Verify.

**kMode\_SSS\_ComputeSharedSecret** = 5

**kMode\_SSS\_Digest** = 6  
Message Digest.

**kMode\_SSS\_Mac** = 7  
Message Authentication Code.

**kMode\_SSS\_HKDF\_ExpandOnly** = 9  
HKDF Expand Only (RFC 5869)

## Enum `sss_policy_type_u`

- Defined in file `_sss_inc_fsl_sss_policy.h`

## Enum Documentation

**enum `sss_policy_type_u`**

Type of policy

*Values:*

**`KPolicy_None`**

No policy applied

**`KPolicy_Session`**

Policy related to session.

See [\*sss\\_policy\\_session\\_u\*](#)

**`KPolicy_Sym_Key`**

Policy related to key.

See [`sss\_policy\_key\_u`](#)

**`KPolicy_Asym_Key`**

**`KPolicy_UserID`**

**`KPolicy_File`**

**`KPolicy_Counter`**

**`KPolicy_PCR`**

**`KPolicy_Common`**

**`KPolicy_Common_PCR_Value`**

## Enum `sss_s05x_session_prop_au8_t`

- Defined in file `_sss_inc_fsl_sss_se05x_types.h`

## Enum Documentation

**enum `sss_s05x_session_prop_au8_t`**

SE050 Properties that can be represented as an array

*Values:*

**`kSSS_SE05x_SessionProp_CertUID`** = *kSSS\_SessionProp\_au8\_Proprietary\_Start* + 1

## Enum sss\_s05x\_session\_prop\_u32\_t

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_types.h

### Enum Documentation

#### enum sss\_s05x\_session\_prop\_u32\_t

SE050 Properties that can be represented as 32bit numbers

*Values:*

**kSSS\_SE05x\_SessionProp\_CertUIDLen** = *kSSS\_SessionProp\_u32\_Optional\_Start* + 1

## Enum sss\_session\_prop\_au8\_t

- Defined in file\_sss\_inc\_fsl\_sss\_api.h

### Enum Documentation

#### enum sss\_session\_prop\_au8\_t

Properties of session that are S32

From 0 to kSSS\_SessionProp\_Optional\_Prop\_Start, around  $2^{24} = 16777215$  Properties are possible.

From 0 to kSSS\_SessionProp\_Optional\_Prop\_Start, around  $2^{24} = 16777215$  Properties are possible.

*Values:*

**kSSS\_SessionProp\_au8\_NA** = 0

Invalid

**kSSS\_SessionProp\_szName**

Name of the product, string

**kSSS\_SessionProp\_UID**

Unique Identifier

**kSSS\_SessionProp\_au8\_Optional\_Start** = 0x00FFFFFFu

Optional Properties Start

**kSSS\_SessionProp\_au8\_Proprietary\_Start** = 0x01FFFFFFu

Proprietary Properties Start

## Enum sss\_session\_prop\_u32\_t

- Defined in file\_sss\_inc\_fsl\_sss\_api.h

## Enum Documentation

### enum sss\_session\_prop\_u32\_t

Properties of session that are U32

From 0 to kSSS\_SessionProp\_Optional\_Prop\_Start, around  $2^{24} = 16777215$  Properties are possible.

From 0 to kSSS\_SessionProp\_Optional\_Prop\_Start, around  $2^{24} = 16777215$  Properties are possible.

*Values:*

**kSSS\_SessionProp\_u32\_NA** = 0

Invalid

**kSSS\_SessionProp\_VerMaj**

Major version

**kSSS\_SessionProp\_VerMin**

Minor Version

**kSSS\_SessionProp\_VerDev**

Development Version

**kSSS\_SessionProp\_UIDLen**

**kSSS\_SessionProp\_u32\_Optional\_Start** = 0x00FFFFFFu

Optional Properties Start

**kSSS\_KeyStoreProp\_FreeMem\_Persistent**

How much persistent memory is free

**kSSS\_KeyStoreProp\_FreeMem\_Transient**

How much transient memory is free

**kSSS\_SessionProp\_u32\_Proprietary\_Start** = 0x01FFFFFFu

Proprietary Properties Start

### Enum sss\_status\_t

- Defined in file `_sss_inc_fsl_sss_api.h`

## Enum Documentation

### enum sss\_status\_t

Status of the SSS APIs

*Values:*

**kStatus\_SSS\_Success** = 0x5a5a5a5au

Operation was successful

**kStatus\_SSS\_Fail** = 0x3c3c0000u

Operation failed

**kStatus\_SSS\_InvalidArgument** = 0x3c3c0001u

Operation not performed because some of the passed parameters were found inappropriate

**kStatus\_SSS\_ResourceBusy** = 0x3c3c0002u

Where the underlying sub-system *supports* multi-threading, Internal status to handle simultaneous access.

This status is not expected to be returned to higher layers.



## Enum sss\_tunnel\_dest\_t

- Defined in file\_sss\_inc\_fsl\_sss\_api.h

## Enum Documentation

### enum sss\_tunnel\_dest\_t

Entity on the other side of the tunnel

*Values:*

**kSSS\_TunnelDest\_None** = 0

Default value

**kSSS\_TunnelType\_Se05x\_Iot\_applet**

SE05X IoT Applet

## Enum sss\_type\_t

- Defined in file\_sss\_inc\_fsl\_sss\_api.h

## Enum Documentation

### enum sss\_type\_t

Cryptographic sub system

*Values:*

**kType\_SSS\_SubSystem\_NONE**

**kType\_SSS\_Software** = ((0x01 << 8) | (0x00))

Software based

**kType\_SSS\_mbedTLS** = ((kType\_SSS\_Software) | (0x01))

**kType\_SSS\_OpenSSL** = ((kType\_SSS\_Software) | (0x02))

**kType\_SSS\_HW** = ((0x02 << 8) | (0x00))

HOST HW Based

**kType\_SSS\_SECO** = ((kType\_SSS\_HW) | (0x01))

**kType\_SSS\_Isolated\_HW** = ((0x04 << 8) | (0x00))

Isolated HW

**kType\_SSS\_Sentinel** = ((kType\_SSS\_Isolated\_HW) | (0x01))

**kType\_SSS\_Sentinel200** = ((kType\_SSS\_Isolated\_HW) | (0x02))

**kType\_SSS\_Sentinel300** = ((kType\_SSS\_Isolated\_HW) | (0x03))

**kType\_SSS\_Sentinel400** = ((kType\_SSS\_Isolated\_HW) | (0x04))

**kType\_SSS\_Sentinel500** = ((kType\_SSS\_Isolated\_HW) | (0x05))

**kType\_SSS\_SecureElement** = ((0x08 << 8) | (0x00))

Secure Element

**kType\_SSS\_SE\_A71CH** = ((kType\_SSS\_SecureElement) | (0x01))

To connect to <https://www.nxp.com/products/:A71CH>

**kType\_SSS\_SE\_A71CL** = ((*kType\_SSS\_SecureElement*) | (0x02))

**kType\_SSS\_SE\_SE05x** = ((*kType\_SSS\_SecureElement*) | (0x03))

To connect to <https://www.nxp.com/products/:SE050>

**kType\_SSS\_SubSystem\_LAST**

## Unions

### Union SE05x\_CryptoModeSubType\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

## Union Documentation

**union SE05x\_CryptoModeSubType\_t**

*#include <se05x\_enums.h>* Cyrypto module subtype

### Public Members

*SE05x\_AeadAlgo\_t* **aead**

In case it's aead

*SE05x\_CipherMode\_t* **cipher**

In case it's cipher

*SE05x\_DigestMode\_t* **digest**

In case it's digest

*SE05x\_MACAlgo\_t* **mac**

In case it's mac

uint8\_t **union\_8bit**

Accessing 8 bit value for APDUs

### Union SE05x\_I2CM\_INS\_type\_t

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_types.h

## Union Documentation

**union SE05x\_I2CM\_INS\_type\_t**

*#include <fsl\_sss\_se05x\_types.h>* Individual entry in array of TLV commands.

## Public Members

*SE05x\_I2CM\_configData\_t* **cfg**

Data Configuration for I2CM

*SE05x\_I2CM\_structuralIssue\_t* **issue**

Used to report error response, not for outgoing command

*SE05x\_I2CM\_readData\_t* **rd**

Read to I2CM from I2C Slave

*SE05x\_I2CM\_securityData\_t* **sec**

Security Configuration for I2CM.

*SE05x\_I2CM\_writeData\_t* **w**

Write From I2CM to I2C Slave.

## Functions

### Function main

- Defined in file\_sss\_ex\_inc\_ex\_sss\_main\_inc.h

### Function Documentation

int **main** (int *argc*, const char \**argv*[])

### Function mbedtls\_ecp\_keypair\_free\_o

- Defined in file\_sss\_plugin\_mbedtls\_sss\_mbedtls.h

### Function Documentation

void **mbedtls\_ecp\_keypair\_free\_o** (mbedtls\_ecp\_keypair \**key*)

This function frees the components of a key pair. Original implementation.

#### Parameters

- *key*: The key pair to free.

### Function mbedtls\_ecp\_tls\_read\_group\_o

- Defined in file\_sss\_plugin\_mbedtls\_sss\_mbedtls.h

## Function Documentation

int **MBEDTLS\_ECP\_TLS\_READ\_GROUP\_O**(mbedtls\_ecp\_group \*grp, const unsigned char \*\*buf, size\_t len)  
same as mbedtls\_ecp\_tls\_read\_group

## Function Se05x\_API\_CheckObjectExists

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

## Function Documentation

smStatus\_t **Se05x\_API\_CheckObjectExists**(pSe05xSession\_t session\_ctx, uint32\_t objectID, SE05x\_Result\_t \*presult)

Se05x\_API\_CheckObjectExists

Check if a Secure Object with a certain identifier exists or not.

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See <a href="#">SE05x_INS_t</a>
P1	P1_DEFAULT	See <a href="#">SE05x_P1_t</a>
P2	P2_EXIST	See <a href="#">SE05x_P2_t</a>
Lc	#(Payload)	
	TLV[TAG_1]	4-byte existing Secure Object identifier.
Le	0x00	

*R-APDU Body*

Value	Description
TLV[TAG_1]	1-byte <a href="#">SE05x_Result_t</a>

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	Data is returned successfully.

## Parameters

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] objectID: object id [1:kSE05x\_TAG\_1]
- [out] presult: [0:kSE05x\_TAG\_1]

## Function Se05x\_API\_CipherFinal

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

## Function Documentation

smStatus\_t **Se05x\_API\_CipherFinal** (pSe05xSession\_t session\_ctx, SE05x\_CryptoObjectID\_t cryptoObjectID, const uint8\_t \*inputData, size\_t inputDataLen, uint8\_t \*outputData, size\_t \*poutputDataLen)

Se05x\_API\_CipherFinal

Finish a sequence of cipher operations.

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	<a href="#">SE05x_INS_t</a>
P1	P1_CIPHER	See <a href="#">SE05x_P1_t</a>
P2	P2_FINAL	See <a href="#">SE05x_P2_t</a>
Lc	#(Payload)	
Payload	TLV[TAG_2]	2-byte Crypto Object identifier
TLV[TAG_3]	Input data	
Le	0x00	Expected returned data.

*R-APDU Body*

Value	Description
TLV[TAG_1]	Output data

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The command is handled successfully.

## Parameters

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] cryptoObjectID: cryptoObjectID [1:kSE05x\_TAG\_2]
- [in] inputData: inputData [2:kSE05x\_TAG\_3]
- [in] inputDataLen: Length of inputData
- [out] outputData: [0:kSE05x\_TAG\_1]
- [inout] poutputDataLen: Length for outputData

## Function Se05x\_API\_CipherInit

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

## Function Documentation

smStatus\_t **Se05x\_API\_CipherInit** (pSe05xSession\_t session\_ctx, uint32\_t objectID, SE05x\_CryptoObjectID\_t cryptoObjectID, const uint8\_t \*IV, size\_t IVLen, const SE05x\_Cipher\_Oper\_t operation)

Se05x\_API\_CipherInit

Initialize a symmetric encryption or decryption. The Crypto Object keeps the state of the cipher operation until it's finalized or deleted. Once the CipherFinal function is executed successfully, the Crypto Object state returns to the state immediately after the previous CipherInit function.

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	SE05x_INS_t
P1	P1_CIPHER	See SE05x_P1_t
P2	P2_ENCRYPT or P2_DECRYPT	See SE05x_P2_t
Lc	#(Payload)	
Payload	TLV[TAG_1]	4-byte identifier of the key object.
	TLV[TAG_2]	2-byte Crypto Object identifier
	TLV[TAG_4]	Initialization Vector [Optional] [Conditional: only when the Crypto Object type equals CC_CIPHER and subtype is not including ECB]
Le	•	

*R-APDU Body*

NA

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The command is handled successfully.

## Parameters

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] objectID: objectID [1:kSE05x\_TAG\_1]
- [in] cryptoObjectID: cryptoObjectID [2:kSE05x\_TAG\_2]
- [in] IV: IV [3:kSE05x\_TAG\_4]
- [in] IVLen: Length of IV
- [in] operation: See SE05x\_Cipher\_Oper\_t

## Function Se05x\_API\_CipherOneShot

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

## Function Documentation

smStatus\_t **Se05x\_API\_CipherOneShot** (pSe05xSession\_t session\_ctx, uint32\_t objectID, SE05x\_CipherMode\_t cipherMode, const uint8\_t \*inputData, size\_t inputDataLen, const uint8\_t \*IV, size\_t IVLen, uint8\_t \*outputData, size\_t \*poutputDataLen, const SE05x\_Cipher\_Oper\_OneShot\_t operation)

Se05x\_API\_CipherOneShot.

Encrypt or decrypt data in one shot mode.

The key object must be either an AES key or DES key.

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	SE05x_INS_t
P1	P1_CIPHER	See SE05x_P1_t
P2	P2_ENCRYPT_ONESHOT or P2_DECRYPT_ONESHOT	See SE05x_P2_t
Lc	#(Payload)	
Pay-load	TLV[TAG_1]	4-byte identifier of the key object.
	TLV[TAG_2]	1-byte CipherMode
	TLV[TAG_3]	Byte array containing input data.
	TLV[TAG_4]	Byte array containing an initialization vector. [Optional] [Conditional: only when the Crypto Object type equals CC_CIPHER and subtype is not including ECB]
Le	0x00	Expecting return data.

*R-APDU Body*

Value	Description
TLV[TAG_1]	Output data

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The command is handled successfully.

**Return** The sm status.

### Parameters

- [in] session\_ctx: The session context
- [in] objectID: The object id

- [in] cipherMode: The cipher mode
- [in] inputData: The input data
- [in] inputDataLen: The input data length
- [in] IV: Initial vector
- [in] IVLen: The iv length
- outputData: The output data
- poutputDataLen: The poutput data length
- [in] operation: The operation

### Function Se05x\_API\_CipherUpdate

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

### Function Documentation

smStatus\_t **Se05x\_API\_CipherUpdate** (pSe05xSession\_t *session\_ctx*, SE05x\_CryptoObjectID\_t *cryptoObjectID*, **const** uint8\_t \**inputData*, size\_t *inputDataLen*, uint8\_t \**outputData*, size\_t \**poutputDataLen*)

Se05x\_API\_CipherUpdate

Update a cipher context.

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_CRYPT0	<a href="#">SE05x_INS_t</a>
P1	P1_CIPHER	See <a href="#">SE05x_P1_t</a>
P2	P2_UPDATE	See <a href="#">SE05x_P2_t</a>
Lc	#(Payload)	
Payload	TLV[TAG_2]	2-byte Crypto Object identifier
TLV[TAG_3]	Byte array containing input data	
Le	0x00	Expecting returned data.

*R-APDU Body*

Value	Description
TLV[TAG_1]	Output data

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The command is handled successfully.

### Parameters

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] cryptoObjectID: cryptoObjectID [1:kSE05x\_TAG\_2]



- [in] inputData: inputData [2:kSE05x\_TAG\_3]
- [in] inputDataLen: Length of inputData
- [out] outputData: [0:kSE05x\_TAG\_1]
- [inout] poutputDataLen: Length for outputData

### Function Se05x\_API\_CloseSession

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

### Function Documentation

smStatus\_t **Se05x\_API\_CloseSession** (pSe05xSession\_t session\_ctx)

Se05x\_API\_CloseSession

Closes a running session.

When a session is closed, it cannot be reopened.

All session parameters are transient.

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See <a href="#">SE05x_INS_t</a>
P1	P1_DEFAULT	See <a href="#">SE05x_P1_t</a>
P2	P2_SESSION_CLOSE	See <a href="#">SE05x_P2_t</a>

*R-APDU Body*

NA

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The session is closed successfully.

### Parameters

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]

### Function Se05x\_API\_CreateCounter

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

## Function Documentation

smStatus\_t **Se05x\_API\_CreateCounter** (pSe05xSession\_t *session\_ctx*, pSe05xPolicy\_t *policy*, uint32\_t *objectID*, uint16\_t *size*)

Se05x\_API\_CreateCounter

Creates a new counter object.

Counters can only be incremented, not decremented.

When a counter reaches its maximum value (e.g., 0xFFFFFFFF for a 4-byte counter), they cannot be incremented again.

An input value (TAG\_3) must always have the same length as the existing counter (if it exists); otherwise the command will return an error.

*Command to Applet*

Field	Value	Description
P1	P1_COUNTER	See <i>SE05x_P1_t</i>
P2	P2_DEFAULT	See <i>SE05x_P2_t</i>
Pay-load	TLV[TAG_POLICY]	Policy array containing the object policy. [Optional: default policy applies] [Conditional: only when the object identifier is not in use yet]
	TLV[TAG_1]	4-byte counter identifier.
	TLV[TAG_2]	2-byte counter size (1 up to 8 bytes). [Conditional: only if object doesn't exist yet and TAG_3 is not given]
	TLV[TAG_3]	Counter value [Optional: - if object doesn't exist: must be present if TAG_2 is not given. - if object exists: if not present, increment by 1. if present, set counter to value.]

*R-APDU Body*

NA

*R-APDU Trailer*

NA

### Parameters

- [in] *session\_ctx*: Session Context [0:kSE05x\_pSession]
- [in] *policy*: policy [1:kSE05x\_TAG\_POLICY]
- [in] *objectID*: object id [2:kSE05x\_TAG\_1]
- [in] *size*: size [3:kSE05x\_TAG\_2]

## Function Se05x\_API\_CreateCryptoObject

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

## Function Documentation

smStatus\_t **Se05x\_API\_CreateCryptoObject** (pSe05xSession\_t *session\_ctx*,  
SE05x\_CryptoObjectID\_t *cryptoObjectID*,  
tID, *SE05x\_CryptoContext\_t* *cryptoContext*,  
*SE05x\_CryptoModeSubType\_t* *subtype*)

Se05x\_API\_CreateCryptoObject

Creates a Crypto Object on the SE05X . Once the Crypto Object is created, it is bound to the user who created the Crypto Object.

A CryptoObject is a 2-byte value consisting of a CryptoContext in MSB and one of the following in LSB:

- DigestMode in case CryptoContext = CC\_DIGEST
- CipherMode in case CryptoContext = CC\_CIPHER
- MACAlgo in case CryptoContext = CC\_SIGNATURE
- AEADMode in case CryptoContext = CC\_AEAD

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_WRITE	See <i>SE05x_INS_t</i>
P1	P1_CRYPTOC	See <i>SE05x_P1_t</i>
P2	P2_DEFAULT	See <i>SE05x_P2_t</i>
Lc	#(Payload)	Payload length
Pay-load	TLV[TAG_1]	2-byte Crypto Object identifier
	TLV[TAG_2]	1-byte <i>SE05x_CryptoObject_t</i>
	TLV[TAG_3]	1-byte Crypto Object subtype, either from DigestModeRef, CipherMode, MACAlgo (depending on TAG_2) or AEADMode.

*R-APDU Body*

NA

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The file is created or updated successfully.

### Parameters

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] cryptoObjectID: cryptoObjectID [1:kSE05x\_TAG\_1]
- [in] cryptoContext: cryptoContext [2:kSE05x\_TAG\_2]
- [in] subtype: 1-byte Crypto Object subtype, either from DigestMode, CipherMode or MACAlgo (depending on TAG\_2). [3:kSE05x\_TAG\_3]

## Function Se05x\_API\_CreateECCurve

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

### Function Documentation

smStatus\_t **Se05x\_API\_CreateECCurve** (pSe05xSession\_t session\_ctx, SE05x\_ECCurve\_t curveID)  
Se05x\_API\_CreateECCurve

Create an EC curve listed in ECCurve.

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_WRITE	See SE05x_INS_t
P1	P1_CURVE	See SE05x_P1_t
P2	P2_CREATE	See SE05x_P2_t
Lc	#(Payload)	
	TLV[TAG_1]	1-byte curve identifier (from SE05x_ECCurve_t).
Le		

*R-APDU Body*

NA

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	Data is returned successfully.

### Parameters

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] curveID: curve id [1:kSE05x\_TAG\_1]

## Function Se05x\_API\_CreateSession

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

### Function Documentation

smStatus\_t **Se05x\_API\_CreateSession** (pSe05xSession\_t session\_ctx, uint32\_t authObjectID, uint8\_t  
\*sessionId, size\_t \*psessionIdLen)

Se05x\_API\_CreateSession

Creates a session on SE05X .

Depending on the authentication object being referenced, a specific method of authentication applies. The response needs to adhere to this authentication method.

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See <a href="#">SE05x_INS_t</a>
P1	P1_DEFAULT	See <a href="#">SE05x_P1_t</a>
P2	P2_SESSION_CREATE	See <a href="#">SE05x_P2_t</a>
Lc	#(Payload)	Payload length.
Payload	TLV[TAG_1]	4-byte authentication object identifier.
Le	0x0A	Expecting TLV with 8-byte session ID.

*R-APDU Body*

Value	Description
TLV[TAG_1]	8-byte session identifier.

*R-APDU Trailer*

SW\_NO\_ERROR:

- The command is handled successfully.

SW\_CONDITIONS\_NOT\_SATISFIED:

- The authenticator does not exist
- The provided input data are incorrect.
- The session is invalid.

**Parameters**

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] authObjectID: auth [1:kSE05x\_TAG\_1]
- [out] sessionId: [0:kSE05x\_TAG\_1]
- [inout] psessionIdLen: Length for sessionId

**Function Se05x\_API\_DeleteAll**

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

**Function Documentation**

smStatus\_t **Se05x\_API\_DeleteAll** (pSe05xSession\_t session\_ctx)  
Se05x\_API\_DeleteAll

Delete all Secure Objects, delete all curves and Crypto Objects. Secure Objects that are trust provisioned by NXP are not deleted (i.e., all objects that have Origin set to ORIGIN\_PROVISIONED, including the objects with reserved object identifiers listed in Object attributes).

This command can only be used from sessions that are authenticated using the credential with index RE-SERVED\_ID\_FACTORY\_RESET.

*Important* : if a secure messaging session is up & running (e.g., AESKey or ECKey session) and the command is sent within this session, the response of the DeleteAll command will not be wrapped (i.e., not encrypted and

no R-MAC), so this will also break down the secure channel protocol (as the session is closed by the DeleteAll command itself).

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See <a href="#">SE05x_INS_t</a>
P1	P1_DEFAULT	See <a href="#">SE05x_P1_t</a>
P2	P2_DELETE_ALL	See <a href="#">SE05x_P2_t</a>
Lc	0x00	

*R-APDU Body*

NA

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	Data is returned successfully.

#### Parameters

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]

### Function Se05x\_API\_DeleteAll\_Iterative

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU.h

#### Function Documentation

smStatus\_t **Se05x\_API\_DeleteAll\_Iterative** (pSe05xSession\_t session\_ctx)  
Se05x\_API\_DeleteAll\_Iterative

Go through each object and delete it individually.

This API does not use the Applet API [Se05x\\_API\\_DeleteAll](#). It does not delete ALL objects and purposefully skips few objects.

Instead, this API uses [Se05x\\_API\\_ReadIDList](#) and [Se05x\\_API\\_ReadCryptoObjectList](#) to first fetch list of objects to host, and **selectively** deletes.

For e.g. It does not kill objects from:

- The range SE05X\_OBJID\_SE05X\_APPLET\_RES\_START to SE05X\_OBJID\_SE05X\_APPLET\_RES\_END. This range is used by applet.
- The range EX\_SSS\_OBJID\_DEMO\_AUTH\_START to EX\_SSS\_OBJID\_DEMO\_AUTH\_END, which is used by middleware DEMOS for authentication.
- And others.

Kindly see the Implementation of is API Se05x\_API\_DeleteAll\_Iterative to see the list of ranges that are skipped.

**Return** The status of API.

**Parameters**

- [in] session\_ctx: Session Context

**Function Se05x\_API\_DeleteCryptoObject**

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

**Function Documentation**

smStatus\_t **Se05x\_API\_DeleteCryptoObject** (pSe05xSession\_t *session\_ctx*,  
SE05x\_CryptoObjectID\_t *cryptoObjectID*)

Se05x\_API\_DeleteCryptoObject

Deletes a Crypto Object on the SE05X .

Note: when a Crypto Object is deleted, the memory (as mentioned in ) is de- allocated, but the transient memory is only freed when de-selecting the applet!

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See <a href="#">SE05x_INS_t</a>
P1	P1_CRYPT0_OBJ	See <a href="#">SE05x_P1_t</a>
P2	P2_DELETE_OBJECT	See <a href="#">SE05x_P2_t</a>
Lc	#(Payload)	Payload length
Payload	TLV[TAG_1]	2-byte Crypto Object identifier

*R-APDU Body*

NA

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The file is created or updated successfully.

**Parameters**

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] cryptoObjectID: cryptoObjectID [1:kSE05x\_TAG\_1]

**Function Se05x\_API\_DeleteECCurve**

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

## Function Documentation

smStatus\_t **Se05x\_API\_DeleteECCurve** (pSe05xSession\_t *session\_ctx*, *SE05x\_ECCurve\_t* *curveID*)  
Se05x\_API\_DeleteECCurve

Deletes an EC curve.

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See <i>SE05x_INS_t</i>
P1	P1_CURVE	See <i>SE05x_P1_t</i>
P2	P2_DELETE_OBJECT	See <i>SE05x_P2_t</i>
Lc	#(Payload)	
	TLV[TAG_1]	1-byte curve identifier (from <i>SE05x_ECCurve_t</i> )

*R-APDU Body*

NA

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	Data is returned successfully.

### Parameters

- [in] *session\_ctx*: Session Context [0:kSE05x\_pSession]
- [in] *curveID*: curve id [1:kSE05x\_TAG\_1]

## Function Se05x\_API\_DeleteSecureObject

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

## Function Documentation

smStatus\_t **Se05x\_API\_DeleteSecureObject** (pSe05xSession\_t *session\_ctx*, uint32\_t *objectID*)  
Se05x\_API\_DeleteSecureObject

Deletes a Secure Object.

If the object origin = ORIGIN\_PROVISIONED, an error will be returned and the object is not deleted.

*Command to Applet*



Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See <i>SE05x_INS_t</i>
P1	P1_DEFAULT	See <i>SE05x_P1_t</i>
P2	P2_DELETE_OBJECT	See <i>SE05x_P2_t</i>
Lc	#(Payload)	
	TLV[TAG_1]	4-byte existing Secure Object identifier.
Le	•	

*R-APDU Body*

NA

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The file is created or updated successfully.

#### Parameters

- [in] `session_ctx`: Session Context [0:kSE05x\_pSession]
- [in] `objectID`: object id [1:kSE05x\_TAG\_1]

#### Function `Se05x_API_DFAuthenticateFirstPart1`

- Defined in file `_hostlib_hostLib_se05x_03_xx_xx_se05x_APDU_apis.h`

#### Function Documentation

`smStatus_t Se05x_API_DFAuthenticateFirstPart1` (`pSe05xSession_t session_ctx`, `uint32_t objectID`, `const uint8_t *inputData`, `size_t inputDataLen`, `uint8_t *outputData`, `size_t *poutputDataLen`)

`Se05x_API_DFAuthenticateFirstPart1`

MIFARE DESFire support

MIFARE DESFire EV2 Key derivation (S-mode). This is limited to AES128 keys only.

The SE05X can be used by a card reader to setup a session where the SE05X stores the master key(s) and the session keys are generated and passed to the host.

The SE05X keeps an internal state of MIFARE DESFire authentication data during authentication setup. This state is fully transient, so it is lost on deselect of the applet.

The MIFARE DESFire state is owned by 1 user at a time; i.e., the user who calls `DFAuthenticateFirstPart1` owns the MIFARE DESFire context until `DFAuthenticateFirstPart1` is called again or until `DFKillAuthentication` is called.

The SE05X can also be used to support a `ChangeKey` command, either supporting `ChangeKey` or `ChangeKeyEV2`. To establish a correct use case, policies need to be applied to the keys to indicate keys can be used for `ChangeKey` or not, etc. (to be detailed)

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_CRYPT0	<i>SE05x_INS_t</i>
P1	P1_DEFAULT	See <i>SE05x_P1_t</i>
P2	P2_AUTH_FIRST_PART1	See <i>SE05x_P2_t</i>
Lc	#(Payload)	
	TLV[TAG_1]	4-byte key identifier.
	TLV[TAG_2]	16-byte encrypted card challenge: E(Kx,RndB)
Le	0x00	

*R-APDU Body*

Value	Description
TLV[TAG_1]	32-byte output data: E(Kx, RandA    RandB')

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The command is handled successfully.

**Parameters**

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] objectID: objectID [1:kSE05x\_TAG\_1]
- [in] inputData: inputData [2:kSE05x\_TAG\_2]
- [in] inputDataLen: Length of inputData
- [out] outputData: [0:kSE05x\_TAG\_1]
- [inout] poutputDataLen: Length for outputData

**Function Se05x\_API\_DFAuthenticateFirstPart2**

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

**Function Documentation**

smStatus\_t **Se05x\_API\_DFAuthenticateFirstPart2** (pSe05xSession\_t session\_ctx, const uint8\_t \*inputData, size\_t inputDataLen, uint8\_t \*outputData, size\_t \*poutputDataLen)

Se05x\_API\_DFAuthenticateFirstPart2

For First part 2, the key identifier is implicitly set to the identifier used for the First authentication. DFAuthenticateFirstPart1 needs to be called before; otherwise an error is returned.

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_CRYPT0	<i>SE05x_INS_t</i>
P1	P1_DEFAULT	See <i>SE05x_P1_t</i>
P2	P2_AUTH_FIRST_PART2	See <i>SE05x_P2_t</i>
Lc	#(Payload)	
	TLV[TAG_1]	32 byte input: E(Kx,TI  RndA'  PDcap2  PCDcap2)
Le	0x00	

*R-APDU Body*

Value	Description
TLV[TAG_1]	12-byte array returning PDcap2  PCDcap2.

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The command is handled successfully.
SW_WRONG_DATA	
SW_CONDITIONS_NOT_SATISFIED	

**Parameters**

- [in] *session\_ctx*: Session Context [0:kSE05x\_pSession]
- [in] *inputData*: *inputData* [1:kSE05x\_TAG\_1]
- [in] *inputDataLen*: Length of *inputData*
- [out] *outputData*: [0:kSE05x\_TAG\_1]
- [inout] *poutputDataLen*: Length for *outputData*

**Function Se05x\_API\_DFAuthenticateNonFirstPart1**

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

**Function Documentation**

smStatus\_t **Se05x\_API\_DFAuthenticateNonFirstPart1** (pSe05xSession\_t *session\_ctx*, uint32\_t *objectID*, **const** uint8\_t \**inputData*, size\_t *inputDataLen*, uint8\_t \**outputData*, size\_t \**poutputDataLen*)

Se05x\_API\_DFAuthenticateNonFirstPart1

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_CRYPT0	<i>SE05x_INS_t</i>
P1	P1_DEFAULT	See <i>SE05x_P1_t</i>
P2	P2_AUTH_NONFIRST_PART1	See <i>SE05x_P2_t</i>
Lc	#{Payload}	
	TLV[TAG_1]	4-byte key identifier.
	TLV[TAG_2]	16-byte encrypted card challenge: E(Kx,RndB)
Le	0x00	

#### *R-APDU Body*

Value	Description
TLV[TAG_1]	32-byte output data: E(Kx, RandA    RandB')

#### *R-APDU Trailer*

SW	Description
SW_NO_ERROR	The command is handled successfully.

#### Parameters

- [in] `session_ctx`: Session Context [0:kSE05x\_pSession]
- [in] `objectID`: objectID [1:kSE05x\_TAG\_1]
- [in] `inputData`: inputData [2:kSE05x\_TAG\_2]
- [in] `inputDataLen`: Length of inputData
- [out] `outputData`: [0:kSE05x\_TAG\_1]
- [inout] `poutputDataLen`: Length for outputData

### Function Se05x\_API\_DFChangeKeyPart1

- Defined in file `hostlib_hostLib_se05x_03_xx_xx_se05x_APDU_apis.h`

#### Function Documentation

`smStatus_t Se05x_API_DFChangeKeyPart1` (`pSe05xSession_t session_ctx`, `uint32_t oldObjectID`, `uint32_t newObjectID`, `uint8_t keySetNr`, `uint8_t keyNoDESFire`, `uint8_t keyVer`, `uint8_t *KeyData`, `size_t *pKeyDataLen`)

Se05x\_API\_DFChangeKeyPart1

The DFChangeKeyPart1 command is supporting the function to change keys on the DESFire PICC. The command generates the cryptogram required to perform such operation.

The new key and, if used, the current (or old) key must be stored in the SE05X and have the POLICY\_OBJ\_ALLOW\_DESFIRE\_AUTHENTICATION associated to execute this command. This means the new PICC key must have been loaded into the SE05X prior to issuing this command.

The 1-byte key set number indicates whether DESFire ChangeKey or DESFire ChangeKeyEV2 is used. When key set equals 0xFF, ChangeKey is used.

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_CRYPT0	<i>SE05x_INS_t</i>
P1	P1_DEFAULT	See <i>SE05x_P1_t</i>
P2	P2_CHANGE_KEY	See <i>SE05x_P2_t</i>
Lc	#(Payload)	
	TLV[TAG_1]	4-byte identifier of the old key. [Optional: if the authentication key is the same as the key to be replaced, this TAG should not be present].
	TLV[TAG_2]	4-byte identifier of the new key.
	TLV[TAG_3]	1-byte key set number [Optional: default = 0xC6]
	TLV[TAG_4]	1-byte DESFire key number to be targeted.
	TLV[TAG_5]	1-byte key version
Le	0x00	

*R-APDU Body*

Value	Description
TLV[TAG_1]	Cryptogram holding key data

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The command is handled successfully.

### Parameters

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] oldObjectID: oldObjectID [1:kSE05x\_TAG\_1]
- [in] newObjectID: newObjectID [2:kSE05x\_TAG\_2]
- [in] keySetNr: keySetNr [3:kSE05x\_TAG\_3]
- [in] keyNoDESFire: keyNoDESFire [4:kSE05x\_TAG\_4]
- [in] keyVer: keyVer [5:kSE05x\_TAG\_5]
- [out] KeyData: [0:kSE05x\_TAG\_1]
- [inout] pKeyDataLen: Length for KeyData

## Function Se05x\_API\_DFChangeKeyPart2

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

### Function Documentation

smStatus\_t **Se05x\_API\_DFChangeKeyPart2** (pSe05xSession\_t *session\_ctx*, **const** uint8\_t \**MAC*, size\_t *MACLen*, uint8\_t \**presult*)

Se05x\_API\_DFChangeKeyPart2

The DFChangeKeyPart2 command verifies the MAC returned by ChangeKey or ChangeKeyEV2. Note that this function only needs to be called if a MAC is returned (which is not the case if the currently authenticated key is changed on the DESFire card).

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_CRYPT0	<a href="#">SE05x_INS_t</a>
P1	P1_DEFAULT	See <a href="#">SE05x_P1_t</a>
P2	P2_CHANGE_KEY_PART2	See <a href="#">SE05x_P2_t</a>
Lc	#(Payload)	
	TLV[TAG_1]	MAC
Le	0x00	

*R-APDU Body*

Value	Description
TLV[TAG_1]	1-byte <a href="#">SE05x_Result_t</a>

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The command is handled successfully.

### Parameters

- [in] *session\_ctx*: Session Context [0:kSE05x\_pSession]
- [in] *MAC*: MAC [1:kSE05x\_TAG\_1]
- [in] *MACLen*: Length of MAC
- [out] *presult*: [0:kSE05x\_TAG\_1]

## Function Se05x\_API\_DFDiversifyKey

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

## Function Documentation

smStatus\_t **Se05x\_API\_DFDiversifyKey** (pSe05xSession\_t *session\_ctx*, uint32\_t *masterKeyID*, uint32\_t *diversifiedKeyID*, **const** uint8\_t \**divInputData*, size\_t *divInputDataLen*)

Se05x\_API\_DFDiversifyKey

Create a Diversified Key. Input is *divInput* 1 up to 31 bytes.

Note that users need to create the diversified key object before calling this function.

Both the master key and the diversified key need the policy POLICY\_OBJ\_ALLOW\_DESFIRE\_AUTHENTICATION to be set.

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_CRYPT0	<a href="#">SE05x_INS_t</a>
P1	P1_DEFAULT	See <a href="#">SE05x_P1_t</a>
P2	P2_DIVERSIFY	See <a href="#">SE05x_P2_t</a>
Lc	#(Payload)	
	TLV[TAG_1]	4-byte master key identifier.
	TLV[TAG_2]	4-byte diversified key identifier.
	TLV[TAG_3]	Byte array containing divInput (up to 31 bytes).
Le		

*R-APDU Body*

NA

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The command is handled successfully.
SW_CONDITIONS_NOT_SATISFIED	No master key found.
	Wrong length for divInput.

## Parameters

- [in] *session\_ctx*: Session Context [0:kSE05x\_pSession]
- [in] *masterKeyID*: masterKeyID [1:kSE05x\_TAG\_1]
- [in] *diversifiedKeyID*: diversifiedKeyID [2:kSE05x\_TAG\_2]
- [in] *divInputData*: divInputData [3:kSE05x\_TAG\_3]
- [in] *divInputDataLen*: Length of divInputData

## Function Se05x\_API\_DFDumpSessionKeys

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

### Function Documentation

smStatus\_t **Se05x\_API\_DFDumpSessionKeys** (pSe05xSession\_t session\_ctx, uint8\_t \*sessionData, size\_t \*psessionDataLen)

Se05x\_API\_DFAuthenticateNonFirstPart2

For NonFirst part 2, the key identifier is implicitly set to the identifier used for the NonFirst part 1 authentication. DFAuthenticateNonFirstPart1 needs to be called before; otherwise an error is returned.

If authentication fails, SW\_WRONG\_DATA will be returned.

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_CRYPT0	<a href="#">SE05x_INS_t</a>
P1	P1_DEFAULT	See <a href="#">SE05x_P1_t</a>
P2	P2_AUTH_NONFIRST_PART2	See <a href="#">SE05x_P2_t</a>
Lc	#(Payload)	
	TLV[TAG_1]	16-byte E(Kx, RndA')
Le	0x00	

*R-APDU Body*

NA

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The command is handled successfully.

@param[in] session\_ctx Session Context [0:kSE05x\_pSession] @param[in] inputData in-putData [1:kSE05x\_TAG\_1] @param[in] inputDataLen Length of inputData

/

smStatus\_t **Se05x\_API\_DFAuthenticateNonFirstPart2**( pSe05xSession\_t session\_ctx, const uint8\_t inputData, size\_t inputDataLen);

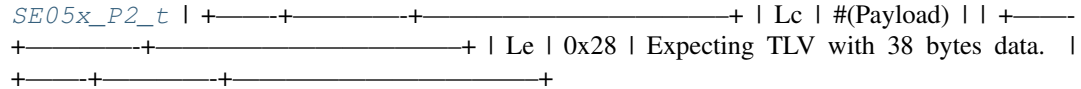
/\* Se05x\_API\_DFDumpSessionKeys

Dump the Transaction Identifier and the session keys to the host.

# Command to Applet

```
verbatim embed:rst:leading-asterisk +-----+-----+-----+-----+ | Field |
Value | Description | +-----+-----+-----+-----+ +-----+
| CLA | 0x80 | | +-----+-----+-----+-----+ | INS | INS_CRYPT0 |
SE05x_INS_t | +-----+-----+-----+-----+ | P1 | P1_DEFAULT | See
SE05x_P1_t | +-----+-----+-----+-----+ | P2 | P2_DUMP_KEY | See
```





*R-APDU Body*

Value	Description
TLV[TAG_1]	38 bytes: KeyID.SesAuthENCKey    KeyID.SesAuthMACKey    TI    Cmd-Ctr

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The command is handled successfully.

### Parameters

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [out] sessionData: 38 bytes: KeyID.SesAuthENCKey || KeyID.SesAuthMACKey || TI || Cmd-Ctr [0:kSE05x\_TAG\_1]
- [inout] psessionDataLen: Length for sessionData

## Function Se05x\_API\_DFKillAuthentication

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

### Function Documentation

smStatus\_t **Se05x\_API\_DFKillAuthentication** (pSe05xSession\_t session\_ctx)  
Se05x\_API\_DFKillAuthentication

DFKillAuthentication invalidates any authentication and clears the internal DESFire state. Keys used as input (master keys or diversified keys) are not touched.

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_CRYPT0	SE05x_INS_t
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_KILL_AUTH	See SE05x_P2_t
Lc	#(Payload)	

*R-APDU Body*

NA

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The command is handled successfully.

### Parameters

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]

## Function Se05x\_API\_DigestFinal

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

## Function Documentation

smStatus\_t **Se05x\_API\_DigestFinal** (pSe05xSession\_t session\_ctx, SE05x\_CryptoObjectID\_t cryptoObjectID, **const** uint8\_t \*inputData, size\_t inputDataLen, uint8\_t \*cmacValue, size\_t \*pcmacValueLen)

Se05x\_API\_DigestFinal

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	See <a href="#">SE05x_INS_t</a>
P1	P1_DEFAULT	See <a href="#">SE05x_P1_t</a>
P2	P2_FINAL	See <a href="#">SE05x_P2_t</a>
Lc	#(Payload)	
	TLV[TAG_2]	2-byte Crypto Object identifier
	TLV[TAG_3]	Data to be encrypted or decrypted.
Le	0x00	Expecting TLV with hash value.

*R-APDU Body*

Value	Description
TLV[TAG_1]	CMAC value

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The hash is created successfully.

## Parameters

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] cryptoObjectID: cryptoObjectID [1:kSE05x\_TAG\_2]
- [in] inputData: inputData [2:kSE05x\_TAG\_3]
- [in] inputDataLen: Length of inputData
- [out] cmacValue: [0:kSE05x\_TAG\_1]
- [inout] pcmacValueLen: Length for cmacValue

## Function Se05x\_API\_DigestInit

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

### Function Documentation

smStatus\_t **Se05x\_API\_DigestInit** (pSe05xSession\_t *session\_ctx*, SE05x\_CryptoObjectID\_t *cryptoObjectID*)

Se05x\_API\_DigestInit

Open a digest operation. The state of the digest operation is kept in the Crypto Object until the Crypto Object is finalized or deleted.

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_CRYPT0	See <a href="#">SE05x_INS_t</a>
P1	P1_DEFAULT	See <a href="#">SE05x_P1_t</a>
P2	P2_INIT	See <a href="#">SE05x_P2_t</a>
Lc	#(Payload)	
	TLV[TAG_2]	2-byte Crypto Object identifier

*R-APDU Body*

NA

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The command is handled successfully.

### Parameters

- [in] *session\_ctx*: Session Context [0:kSE05x\_pSession]
- [in] *cryptoObjectID*: cryptoObjectID [1:kSE05x\_TAG\_2]

## Function Se05x\_API\_DigestOneShot

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

### Function Documentation

smStatus\_t **Se05x\_API\_DigestOneShot** (pSe05xSession\_t *session\_ctx*, uint8\_t *digestMode*, const uint8\_t \**inputData*, size\_t *inputDataLen*, uint8\_t \**hashValue*, size\_t \**hashValueLen*)

Se05x\_API\_DigestOneShot

Performs a hash operation in one shot (without context).

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	See <i>SE05x_INS_t</i>
P1	P1_DEFAULT	See <i>SE05x_P1_t</i>
P2	P2_ONESHOT	See <i>SE05x_P2_t</i>
Lc	#(Payload)	
	TLV[TAG_1]	1-byte DigestMode (except DIGEST_NO_HASH)
	TLV[TAG_2]	Data to hash.
Le	0x00	TLV expecting hash value

#### *R-APDU Body*

Value	Description
TLV[TAG_1]	Hash value.

#### *R-APDU Trailer*

SW	Description
SW_NO_ERROR	The hash is created successfully.

#### Parameters

- [in] `session_ctx`: Session Context [0:kSE05x\_pSession]
- [in] `digestMode`: digestMode [1:kSE05x\_TAG\_1]
- [in] `inputData`: inputData [2:kSE05x\_TAG\_2]
- [in] `inputDataLen`: Length of inputData
- [out] `hashValue`: [0:kSE05x\_TAG\_1]
- [inout] `phashValueLen`: Length for hashValue

### Function Se05x\_API\_DigestUpdate

- Defined in file `hostlib_hostLib_se05x_03_xx_xx_se05x_APDU_apis.h`

#### Function Documentation

`smStatus_t Se05x_API_DigestUpdate` (`pSe05xSession_t session_ctx`, `SE05x_CryptoObjectID_t cryptoObjectID`, `const uint8_t *inputData`, `size_t inputDataLen`)

`Se05x_API_DigestUpdate`

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	See <i>SE05x_INS_t</i>
P1	P1_DEFAULT	See <i>SE05x_P1_t</i>
P2	P2_UPDATE	See <i>SE05x_P2_t</i>
Lc	#(Payload)	
	TLV[TAG_2]	2-byte Crypto Object identifier
	TLV[TAG_3]	Data to be hashed.
Le		

*R-APDU Body*

NA

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The command is handled successfully.

#### Parameters

- [in] `session_ctx`: Session Context [0:kSE05x\_pSession]
- [in] `cryptoObjectID`: cryptoObjectID [1:kSE05x\_TAG\_2]
- [in] `inputData`: inputData [2:kSE05x\_TAG\_3]
- [in] `inputDataLen`: Length of inputData

#### Function Se05x\_API\_EC\_CurveGetId

- Defined in file `hostlib_hostLib_se05x_03_xx_xx_se05x_APDU.h`

#### Function Documentation

`smStatus_t Se05x_API_EC_CurveGetId (pSe05xSession_t session_ctx, uint32_t objectID, SE05x_ECCurve_t *pcurveId)`

Get the Curve ID for existing Key.

This API is functionally same as *Se05x\_API\_GetECCurveId* but uses *SE05x\_ECCurve\_t* as a type instead of `uint8_t`.

**Return** The sm status.

#### Parameters

- [in] `session_ctx`: The session context
- [in] `objectID`: The object id
- `pcurveId`: The pcurve identifier

## Function Se05x\_API\_ECDAASign

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

### Function Documentation

```
smStatus_t Se05x_API_ECDAASign (pSe05xSession_t session_ctx, uint32_t objectID,
 SE05x_ECDAASignatureAlgo_t ecdaaSignAlgo, const uint8_t
 *inputData, size_t inputDataLen, const uint8_t *randomData,
 size_t randomDataLen, uint8_t *signature, size_t *psignatureLen)
```

Se05x\_API\_ECDAASign

The ECDAASign command signs external data using the indicated key pair or private key. This is performed according to ECDA. The generated signature is:

- $r = \text{random} \bmod n$
- $s = (r + T \cdot d) \bmod n$  where  $d$  is the private key

The ECDAASignatureAlgo indicates the applied algorithm.

This APDU command should be used with a key identifier linked to TPM\_ECC\_BN\_P256 curve.

*Note:* The applet allows the random input to be 32 bytes of zeroes; the user must take care that this is not considered as valid input. Only input in the interval  $[1, n-1]$  must be considered as valid.

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	SE05x_INS_t
P1	P1_SIGNATURE	See SE05x_P1_t
P2	P2_SIGN	See SE05x_P2_t
Lc	#(Payload)	
	TLV[TAG_1]	4-byte identifier of EC key pair or private key.
	TLV[TAG_2]	1-byte ECDAASignatureAlgo
	TLV[TAG_3]	T = 32-byte array containing hashed input data.
	TLV[TAG_4]	r = 32-byte array containing random data, must be in the interval $[1, n-1]$ where $n$ is the order of the curve.
Le	0x00	Expecting signature

*R-APDU Body*

Value	Description
TLV[TAG_1]	ECDSA Signature (r concatenated with s).

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The command is handled successfully.

### Parameters

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]

- [in] objectID: objectID [1:kSE05x\_TAG\_1]
- [in] ecdaaSignAlgo: ecdaaSignAlgo [2:kSE05x\_TAG\_2]
- [in] inputData: inputData [3:kSE05x\_TAG\_3]
- [in] inputDataLen: Length of inputData
- [in] randomData: randomData [4:kSE05x\_TAG\_4]
- [in] randomDataLen: Length of randomData
- [out] signature: [0:kSE05x\_TAG\_1]
- [inout] psignatureLen: Length for signature

### Function Se05x\_API\_ECDHGenerateSharedSecret

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

### Function Documentation

smStatus\_t **Se05x\_API\_ECDHGenerateSharedSecret** (pSe05xSession\_t session\_ctx, uint32\_t objectID, const uint8\_t \*pubKey, size\_t pubKeyLen, uint8\_t \*sharedSecret, size\_t \*psharedSecretLen)

Se05x\_API\_ECDHGenerateSharedSecret

The ECDHGenerateSharedSecret command generates a shared secret ECC point on the curve using an EC private key on SE05X and an external public key provided by the caller. The output shared secret is returned to the caller.

All curves from ECCurve are supported, except ECC\_ED\_25519.

Note that ECDHGenerateSharedSecret commands with EC keys using curve ID\_ECC\_MONT\_DH\_25519 or ID\_ECC\_MONT\_DH\_448 cause NVM write operations for each call. This is not the case for the other curves.

When CONFIG\_FIPS\_MODE\_DISABLED is not set, this function will always return SW\_CONDITIONS\_NOT\_SATISFIED.

The shared secret can only be received when the Secure Object containing the key pair or private key (TLV[TAG\_1]) does not contain the policy POLICY\_OBJ\_FORBID\_DERIVED\_OUTPUT. If that is the case, the user must provide TLV[TAG\_7] to store the shared secret in an HMACKey object. The user is responsible to assign the correct size of the HMACKey object: this must equal the size of the shared secret exactly.

On applet 4.4.0, the policy POLICY\_OBJ\_FORBID\_DERIVED\_OUTPUT is not yet verified for this function. It will always be allowed.

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_CRYPT0	<i>SE05x_INS_t</i>
P1	P1_EC	See <i>SE05x_P1_t</i>
P2	P2_DH	See <i>SE05x_P2_t</i>
Lc	#{Payload}	
Payload	TLV[TAG_1]	4-byte identifier of the key pair or private key.
TLV[TAG_2]	External public key (see ECKKeyRef).	
TLV[TAG_7]	4-byte HMACKey identifier to store output. [Optional]	
Le	0x00	Expected shared secret length.

*R-APDU Body*

Value	Description
TLV[TAG_1]	The returned shared secret. [Conditional: only when the input does not contain TLV[TAG_7].]

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The command is handled successfully.

**Parameters**

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] objectID: objectID [1:kSE05x\_TAG\_1]
- [in] pubKey: pubKey [2:kSE05x\_TAG\_2]
- [in] pubKeyLen: Length of pubKey
- [out] sharedSecret: [0:kSE05x\_TAG\_1]
- [inout] psharedSecretLen: Length for sharedSecret

**Function Se05x\_API\_ECDSASign**

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

**Function Documentation**

```
smStatus_t Se05x_API_ECDSASign (pSe05xSession_t session_ctx, uint32_t objectID,
 SE05x_ECSignatureAlgo_t ecSignAlgo, const uint8_t *inputData,
 size_t inputDataLen, uint8_t *signature, size_t *psignatureLen)
```

Se05x\_API\_ECDSASign

The ECDSASign command signs external data using the indicated key pair or private key.

The ECSignatureAlgo indicates the ECDSA algorithm that is used, but the hashing of data always must be done on the host. E.g., if ECSignatureAlgo = SIG\_ECDSA\_SHA256, the user must have applied SHA256 on the input data already.



The user must take care of providing the correct input length; i.e., the data input length (TLV[TAG\_3]) must match the digest indicated in the signature algorithm (TLV[TAG\_2]).

In any case, the APDU payload must be smaller than MAX\_APDU\_PAYLOAD\_LENGTH.

This is performed according to the ECDSA algorithm as specified in [ANSI X9.62]. The signature (a sequence of two integers ‘r’ and ‘s’) as returned in the response adheres to the ASN.1 DER encoded formatting rules for integers.

#### *Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	<i>SE05x_INS_t</i>
P1	P1_SIGNATURE	See <i>SE05x_P1_t</i>
P2	P2_SIGN	See <i>SE05x_P2_t</i>
Lc	#(Payload)	
	TLV[TAG_1]	4-byte identifier of EC key pair or private key.
	TLV[TAG_2]	1-byte ECSignatureAlgo.
	TLV[TAG_3]	Byte array containing input data.
Le	0x00	Expecting ASN.1 signature

#### *R-APDU Body*

Value	Description
TLV[TAG_1]	ECDSA Signature in ASN.1 format.

#### *R-APDU Trailer*

SW	Description
SW_NO_ERROR	The command is handled successfully.

#### Parameters

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] objectID: objectID [1:kSE05x\_TAG\_1]
- [in] ecSignAlgo: ecSignAlgo [2:kSE05x\_TAG\_2]
- [in] inputData: inputData [3:kSE05x\_TAG\_3]
- [in] inputDataLen: Length of inputData
- [out] signature: [0:kSE05x\_TAG\_1]
- [inout] psignatureLen: Length for signature

## Function Se05x\_API\_ECDSAVerify

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

### Function Documentation

smStatus\_t **Se05x\_API\_ECDSAVerify** (pSe05xSession\_t session\_ctx, uint32\_t objectID, SE05x\_ECSignatureAlgo\_t ecSignAlgo, const uint8\_t \*inputData, size\_t inputDataLen, const uint8\_t \*signature, size\_t signatureLen, SE05x\_Result\_t \*presult)

#### Se05x\_API\_ECDSAVerify

The ECDSAVerify command verifies whether the signature is correct for a given (hashed) data input using an EC public key or EC key pair's public key.

The ECSignatureAlgo indicates the ECDSA algorithm that is used, but the hashing of data must always be done on the host. E.g., if ECSignatureAlgo = SIG\_ECDSA\_SHA256, the user must have applied SHA256 on the input data already.

The key cannot be passed externally to the command directly. In case users want to use the command to verify signatures using different public keys or the public key value regularly changes, the user should create a transient key object to which the key value is written and then the identifier of that transient secure object can be used by this ECDSAVerify command.

#### Command to Applet

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	SE05x_INS_t
P1	P1_SIGNATURE	See SE05x_P1_t
P2	P2_VERIFY	See SE05x_P2_t
Lc	#(Payload)	
	TLV[TAG_1]	4-byte identifier of the key pair or public key.
	TLV[TAG_2]	1-byte ECSignatureAlgo.
	TLV[TAG_3]	Byte array containing ASN.1 signature
	TLV[TAG_5]	Byte array containing hashed data to compare.
Le	0x03	Expecting TLV with SE05x_Result_t

#### R-APDU Body

Value	Description
TLV[TAG_1]	Result of the signature verification (SE05x_Result_t).

#### R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.
SW_CONDITIONS_NOT_SATISFIED	Incorrect data

#### Parameters

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] objectID: objectID [1:kSE05x\_TAG\_1]

- [in] ecSignAlgo: ecSignAlgo [2:kSE05x\_TAG\_2]
- [in] inputData: inputData [3:kSE05x\_TAG\_3]
- [in] inputDataLen: Length of inputData
- [in] signature: signature [4:kSE05x\_TAG\_5]
- [in] signatureLen: Length of signature
- [out] presult: [0:kSE05x\_TAG\_1]

## Function Se05x\_API\_EdDSASign

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

## Function Documentation

smStatus\_t **Se05x\_API\_EdDSASign** (pSe05xSession\_t *session\_ctx*, uint32\_t *objectID*, *SE05x\_EDSignatureAlgo\_t* *edSignAlgo*, **const** uint8\_t \**inputData*, size\_t *inputDataLen*, uint8\_t \**signature*, size\_t \**psignatureLen*)

Se05x\_API\_EdDSASign

The EdDSASign command signs external data using the indicated key pair or private key (using a Twisted Edwards curve). This is performed according to the EdDSA algorithm as specified in [RFC8032].

The input data need to be the plain data (not hashed).

The signature as returned in the response is a 64-byte array, being the concatenation of the signature r and s component (without leading zeroes for sign indication).

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_CRYPT0	<i>SE05x_INS_t</i>
P1	P1_SIGNATURE	See <i>SE05x_P1_t</i>
P2	P2_SIGN	See <i>SE05x_P2_t</i>
Lc	#(Payload)	
	TLV[TAG_1]	4-byte identifier of EC key pair or private key.
	TLV[TAG_2]	1-byte EDSignatureAlgo
	TLV[TAG_3]	Byte array containing plain input data.
Le	0x00	Expecting signature

*R-APDU Body*

Value	Description
TLV[TAG_1]	EdDSA Signature (r concatenated with s).

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The command is handled successfully.

## Parameters

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] objectID: objectID [1:kSE05x\_TAG\_1]
- [in] edSignAlgo: edSignAlgo [2:kSE05x\_TAG\_2]
- [in] inputData: inputData [3:kSE05x\_TAG\_3]
- [in] inputDataLen: Length of inputData
- [out] signature: [0:kSE05x\_TAG\_1]
- [inout] psignatureLen: Length for signature

### Function Se05x\_API\_EdDSAVerify

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

### Function Documentation

smStatus\_t **Se05x\_API\_EdDSAVerify** (pSe05xSession\_t session\_ctx, uint32\_t objectID, SE05x\_EDSignatureAlgo\_t edSignAlgo, const uint8\_t \*inputData, size\_t inputDataLen, const uint8\_t \*signature, size\_t signatureLen, SE05x\_Result\_t \*presult)

Se05x\_API\_EdDSAVerify

The EdDSAVerify command verifies whether the signature is correct for a given data input (hashed using SHA512) using an EC public key or EC key pair's public key. The signature needs to be given as concatenation of r and s.

The data needs to be compared with the plain message without being hashed.

*Note* : See chapter 7 for correct byte order as both r and s need to be byte swapped.

This is performed according to the EdDSA algorithm as specified in [RFC8032].

The key cannot be passed externally to the command directly. In case users want to use the command to verify signatures using different public keys or the public key value regularly changes, the user should create a transient key object to which the key value is written and then the identifier of that transient secure object can be used by this EdDSAVerify command.

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	SE05x_INS_t
P1	P1_SIGNATURE	See SE05x_P1_t
P2	P2_VERIFY	See SE05x_P2_t
Lc	#(Payload)	
	TLV[TAG_1]	4-byte identifier of the key pair or public key.
	TLV[TAG_2]	1-byte EDSignatureAlgoRef.
	TLV[TAG_3]	64-byte array containing the signature (concatenation of r and s).
	TLV[TAG_5]	Byte array containing plain data to compare.
Le	0x03	Expecting TLV with SE05x_Result_t

*R-APDU Body*

Value	Description
TLV[TAG_1]	Result of the signature verification ( <i>SE05x_Result_t</i> ).

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The command is handled successfully.
SW_CONDITIONS_NOT_SATISFIED	Incorrect data

**Parameters**

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] objectID: objectID [1:kSE05x\_TAG\_1]
- [in] edSignAlgo: edSignAlgo [2:kSE05x\_TAG\_2]
- [in] inputData: inputData [3:kSE05x\_TAG\_3]
- [in] inputDataLen: Length of inputData
- [in] signature: signature [4:kSE05x\_TAG\_5]
- [in] signatureLen: Length of signature
- [out] presult: [0:kSE05x\_TAG\_1]

**Function Se05x\_API\_ExchangeSessionData**

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

**Function Documentation**

smStatus\_t **Se05x\_API\_ExchangeSessionData** (pSe05xSession\_t session\_ctx, pSe05xPolicy\_t policy)  
 Se05x\_API\_ExchangeSessionData

Sets session policies for the current session.

*Command to Applet*

Field	Value	Description
CLA	0x80 or 0x84	•
INS	INS_MGMT	See <i>SE05x_INS_t</i>
P1	P1_DEFAULT	See <i>SE05x_P1_t</i>
P2	P2_SESSION_POLICY	See P2
Lc	#(Payload)	Payload length.
Payload	TLV[TAG_1]	Session policies
	C-MAC	If applicable
Le	0x00	•

*R-APDU Body*

Value	Description
R-MAC	Optional, depending on established security level

SW	Description
SW_NO_ERROR	The command is handled successfully.
SW_CONDITIONS_NOT_SATISFIED	Invalid policies

### Parameters

- [in] `session_ctx`: Session Context [0:kSE05x\_pSession]
- [in] `policy`: Check pdf [1:kSE05x\_TAG\_1]

### Function Se05x\_API\_ExportObject

- Defined in file `hostlib_hostLib_se05x_03_xx_xx_se05x_APDU_apis.h`

### Function Documentation

`smStatus_t Se05x_API_ExportObject` (`pSe05xSession_t session_ctx`, `uint32_t objectID`, `SE05x_RSAPKeyComponent_t rsaKeyComp`, `uint8_t *data`, `size_t *pdataLen`)

`Se05x_API_ExportObject`

Reads a transient Secure Object from SE05X.

Secure Objects can be serialized so the Secure Object can be represented as a byte array. The byte array contains all attributes of the Secure Object, as well as the value (including the secret part!) of the object.

The purpose of the serialization is to be able to allow export and import of Secure Objects. Serialized Secure Objects can be reconstructed so they can be used as a (normal) Secure Object. Any operation like key or file management and crypto operation can only be done on a deserialized Secure Object.

Users can export transient Secure Objects to a non-trusted environment (e.g., host controller). The object must be AESKey, DESKey, RSAKey or ECCKey.

Exported credentials are always encrypted and MAC'ed.

The following steps are taken:

- The secure element holds a randomly generated persistent 256-bit AES cipher and an 128-bit AES CMAC key. Both keys do not require user interaction, they are internal to the SE05X .
- A Secure Object that is identified for export is serialized. This means the key value as well as all Secure Object attributes are stored as byte array (see Object attributes for attribute details).
- The serialized Secure Object is encrypted using AES CBC (no padding) and using the default IV.
- A CMAC is applied to the serialized Secure Object + metadata using the AES CMAC key.
- The byte array is exported.

An object may only be imported into the store if the SecureObject ID and type are the same as the exported object. Therefore, it is not possible to import if the corresponding object in the applet has been deleted.

NOTES:

- The exported object is not deleted automatically.
- The timestamp has a 100msec granularity, so it is possible to export multiple times with the same timestamp. The freshness (user input) should avoid duplicate attestation results as the user has to provide different freshness input.

#### *Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_READ	See <i>SE05x_INS_t</i> .
P1	P1_DEFAULT	See <i>SE05x_P1_t</i>
P2	P2_EXPORT	See <i>SE05x_P2_t</i>
Lc	#(Payload)	Payload Length.
	TLV[TAG_1]	4-byte object identifier
	TLV[TAG_2]	1-byte <i>SE05x_RSAPublicKeyComponent_t</i> (only applies to Secure Objects of type RSAPublicKey).
Le	0x00	

#### *R-APDU Body*

Value	Description
TLV[TAG_1]	Byte array containing exported Secure Object data.

#### *R-APDU Trailer*

SW	Description
SW_NO_ERROR	The file is created or updated successfully.

#### Parameters

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] objectID: object id [1:kSE05x\_TAG\_1]
- [in] rsaKeyComp: rsaKeyComp [2:kSE05x\_TAG\_2]
- [out] data: [0:kSE05x\_TAG\_1]
- [inout] pdataLen: Length for data

#### Function Se05x\_API\_GetECCurveId

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

## Function Documentation

smStatus\_t **Se05x\_API\_GetECCurveId**(pSe05xSession\_t session\_ctx, uint32\_t objectID, uint8\_t \*pcurveId)

Se05x\_API\_GetECCurveId

Get the curve associated with an EC key.

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_READ	See <a href="#">SE05x_INS_t</a>
P1	P1_CURVE	See <a href="#">SE05x_P1_t</a>
P2	P2_ID	See <a href="#">SE05x_P2_t</a>
Lc	#{Payload}	
Payload	TLV[TAG_1]	4-byte identifier
Le	0x00	

*R-APDU Body*

Value	Description
TLV[TAG_1]	1-byte curve identifier (from <a href="#">SE05x_ECCurve_t</a> )

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	Data is returned successfully.

### Parameters

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] objectID: object id [1:kSE05x\_TAG\_1]
- [out] pcurveId: [0:kSE05x\_TAG\_1]

## Function Se05x\_API\_GetFreeMemory

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

## Function Documentation

smStatus\_t **Se05x\_API\_GetFreeMemory**(pSe05xSession\_t session\_ctx, [SE05x\\_MemoryType\\_t](#) memoryType, uint16\_t \*pfreeMem)

Se05x\_API\_GetFreeMemory

Gets the amount of free memory. MemoryType indicates the type of memory.

The result indicates the amount of free memory. Note that behavior of the function might not be fully linear and can have a granularity of 16 bytes where the applet will typically report the “worst case” amount. For example, when allocating 2 bytes a time, the first report will show 16 bytes being allocated, which remains the same for the next 7 allocations of 2 bytes.



*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See <i>SE05x_INS_t</i>
P1	P1_DEFAULT	See <i>SE05x_P1_t</i>
P2	P2_MEMORY	See <i>SE05x_P2_t</i>
Lc	#(Payload)	
	TLV[TAG_1]	<i>SE05x_MemTyp_t</i>
Le	0x04	Expecting TLV with 2-byte data.

*R-APDU Body*

Value	Description
TLV[TAG_1]	12 bytes indicating the amount of free memory of the requested memory type. 0x7FFF as response means at least 32768 bytes are available.

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	Data is returned successfully.

**Return** The sm status.

**Parameters**

- [in] *session\_ctx*: The session context
- [in] *memoryType*: The memory type
- *pfreeMem*: The pfree memory

## Function Se05x\_API\_GetRandom

- Defined in file *hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h*

## Function Documentation

smStatus\_t **Se05x\_API\_GetRandom** (pSe05xSession\_t *session\_ctx*, uint16\_t *size*, uint8\_t \**randomData*, size\_t \**prandomDataLen*)

Se05x\_API\_GetRandom

Gets random data from the SE05X .

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See <a href="#">SE05x_INS_t</a>
P1	P1_DEFAULT	See <a href="#">SE05x_P1_t</a>
P2	P2_RANDOM	See <a href="#">SE05x_P2_t</a>
Lc	#(Payload)	
	TLV[TAG_1]	2-byte requested size.
Le	0x00	Expecting random data

#### *R-APDU Body*

Value	Description
TLV[TAG_1]	Random data.

#### *R-APDU Trailer*

SW	Description
SW_NO_ERROR	Data is returned successfully.

**Return** The sm status.

#### **Parameters**

- [in] session\_ctx: The session context
- [in] size: The size
- randomData: The random data
- prandomDataLen: The prandom data length

### **Function Se05x\_API\_GetTimestamp**

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

#### **Function Documentation**

smStatus\_t **Se05x\_API\_GetTimestamp** (pSe05xSession\_t session\_ctx, SE05x\_TimeStamp\_t \*ptimeStamp)

Se05x\_API\_GetTimestamp

Gets a monotonic counter value (time stamp) from the operating system of the device (both persistent and transient part). See TimestampFunctionality for details on the timestamps.

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See <a href="#">SE05x_INS_t</a>
P1	P1_DEFAULT	See <a href="#">SE05x_P1_t</a>
P2	P2_TIME	See <a href="#">SE05x_P2_t</a>
Lc	#(Payload)	
Le	0x2C	Expecting TLV with timestamp.

*R-APDU Body*

Value	Description
TLV[TAG_1]	TLV containing a 12-byte operating system timestamp.

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	Data is returned successfully.

**Return** The sm status.

**Parameters**

- [in] `session_ctx`: The session context
- `ptimeStamp`: The ptime stamp

**Function Se05x\_API\_GetVersion**

- Defined in file `_hostlib_hostLib_se05x_03_xx_xx_se05x_APDU_apis.h`

**Function Documentation**

`smStatus_t Se05x_API_GetVersion` (`pSe05xSession_t session_ctx`, `uint8_t *pappletVersion`, `size_t *appletVersionLen`)

`Se05x_API_GetVersion`

Gets the applet version information.

This will return 7-byte VersionInfo (including major, minor and patch version of the applet, supported applet features and secure box version).

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See <a href="#">SE05x_INS_t</a>
P1	P1_DEFAULT	See <a href="#">SE05x_P1_t</a>
P2	P2_VERSION or P2_VERSION_EXT	See <a href="#">SE05x_P2_t</a>
Lc	#(Payload)	
Le	0x00	Expecting TLV with 7-byte data (when P2 = P2_VERSION) or a TLV with 37 byte data (when P2= P2_VERSION_EXT).

*R-APDU Body*

Value	Description
TLV[TAG]	17-byte VersionInfoRef (if P2 = P2_VERSION) or 7-byte VersionInfo followed by 30 bytes extendedFeatureBits (if P2 = P2_VERSION_EXT)

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	Data is returned successfully.

**Return** The sm status.

#### Parameters

- [in] `session_ctx`: The session context
- `pappletVersion`: The papplet version
- `appletVersionLen`: The applet version length

### Function Se05x\_API\_HKDF

- Defined in file `hostlib_hostLib_se05x_03_xx_xx_se05x_APDU_apis.h`

### Function Documentation

`smStatus_t Se05x_API_HKDF` (`pSe05xSession_t session_ctx`, `uint32_t hmacID`, `SE05x_DigestMode_t digestMode`, `const uint8_t *salt`, `size_t saltLen`, `const uint8_t *info`, `size_t infoLen`, `uint16_t deriveDataLen`, `uint8_t *hkdfOutput`, `size_t *phkdfOutputLen`)

`Se05x_API_HKDF`

Note that this KDF is equal to the KDF in Feedback Mode described in [NIST SP800-108] with the PRF being HMAC with SHA256 and with an 8-bit counter at the end of the iteration variable.

The full HKDF algorithm is executed, i.e. Extract-And-Expand.

The caller must provide a salt length (0 up to 64 bytes). If salt length equals 0 or salt is not provided as input, the default salt will be used.

The output of the HKDF functions can be either:

- send back to the caller => *precondition* : none of the input Secure Objects -if present- shall have a policy `POLICY_OBJ_FORBID_DERIVED_OUTPUT` set.
- be stored in a Secure Object => *precondition* : the Secure Object must be created upfront and the size must exactly match the expected length.

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_CRYPT0	<i>SE05x_INS_t</i>
P1	P1_DEFAULT	See <i>SE05x_P1_t</i>
P2	P2_HKDF	See <i>SE05x_P2_t</i>
Lc	#(Payload)	
Payload	TLV[TAG_1]	4-byte HMACKey identifier (= IKM)
TLV[TAG_2]	2-byte DigestMode (except DIGEST_NO_HASH)	
TLV[TAG_3]	3-byte array (0-64 bytes) containing salt. [Optional] [Conditional: only when TLV[TAG_6] is absent.]	
TLV[TAG_4]	4-byte info: The context and information to apply (1 to 80 bytes). [Optional]	
TLV[TAG_5]	2-byte requested length (L): 1 up to MAX_APDU_PAYLOAD_LENGTH	
TLV[TAG_6]	6-byte HMACKey identifier containing salt. [Optional] [Conditional: only when TLV[TAG_3] is absent]	
TLV[TAG_7]	7-byte HMACKey identifier to store output. [Optional]	
Le	0x00	

*R-APDU Body*

Value	Description
TLV[TAG_1]	HKDF output. [Conditional: only when the input does not contain TLV[TAG-7]]

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The HKDF is executed successfully.

**Parameters**

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] hmacID: hmacID [1:kSE05x\_TAG\_1]
- [in] digestMode: digestMode [2:kSE05x\_TAG\_2]
- [in] salt: salt [3:kSE05x\_TAG\_3]
- [in] saltLen: Length of salt
- [in] info: info [4:kSE05x\_TAG\_4]
- [in] infoLen: Length of info
- [in] deriveDataLen: 2-byte requested length (L) [5:kSE05x\_TAG\_5]
- [out] hkdfOutput: [0:kSE05x\_TAG\_1]
- [inout] phkdfOutputLen: Length for hkdfOutput

## Function Se05x\_API\_I2CM\_ExecuteCommandSet

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

## Function Documentation

smStatus\_t **Se05x\_API\_I2CM\_ExecuteCommandSet** (pSe05xSession\_t session\_ctx, **const** uint8\_t \*inputData, size\_t inputDataLen, uint32\_t attestationID, uint8\_t attestationAlgo, uint8\_t \*response, size\_t \*presponseLen, SE05x\_TimeStamp\_t \*ptimeStamp, uint8\_t \*freshness, size\_t \*pfresh-nessLen, uint8\_t \*chipId, size\_t \*pchipIdLen, uint8\_t \*signature, size\_t \*psignatureLen, uint8\_t \*randomAttst, size\_t randomAttstLen)

Se05x\_API\_I2CM\_ExecuteCommandSet

Execute one or multiple I2C commands in master mode. Execution is conditional to the presence of the authentication object identified by RESERVED\_ID\_I2CM\_ACCESS. If the credential is not present in the eSE, access is allowed in general. Otherwise, a session shall be established before executing this command. In this case, the I2CM\_ExecuteCommandSet command shall be sent within the mentioned session.

The I2C command set is constructed as a sequence of instructions described in with the following rules:

- The length should be limited to MAX\_I2CM\_COMMAND\_LENGTH.
- The data to be read cannot exceed MAX\_I2CM\_COMMAND\_LENGTH, including protocol overhead.

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_CRYPT	See <a href="#">SE05x_INS_t</a> , in addition to INS_CRYPT, users can set the INS_ATTEST flag. In that case, attestation applies.
P1	P1_DEFAULT	See <a href="#">SE05x_P1_t</a>
P2	P2_I2CM	See <a href="#">SE05x_P2_t</a>
Lc	#(Pay-load)	
	TLV[TAG_1]	Byte array containing I2C Command set as TLV array.
	TLV[TAG_2]	4-byte attestation object identifier. [Optional] [Conditional: only when INS_ATTEST is set]
	TLV[TAG_3]	1-byte <a href="#">SE05x_AttestationAlgo_t</a> [Optional] [Conditional: only when INS_ATTEST is set]
	TLV[TAG_7]	16-byte freshness random [Optional] [Conditional: only when INS_ATTEST is set]
Le	0x00	Expecting TLV with return data.

*R-APDU Body*

Value	Description
TLV[ <del>TAG_1</del> TAG_2]	Read response, a bytestring containing a sequence of: * CONFIGURE (0x01), followed by 1 byte of return code (0x5A = SUCCESS). * WRITE (0x03), followed by 1 byte of return code * READ (0x04), followed by - Length: 2 bytes in big endian encoded without TLV length encoding - Read bytes * 0xFF followed by the error return code in case of a structural error of the incoming buffer (too long, for example)
TLV[ <del>TAG_3</del> TAG_3]	containing 12-byte timestamp
TLV[ <del>TAG_4</del> TAG_4]	containing 16-byte freshness (random)
TLV[ <del>TAG_5</del> TAG_5]	containing 18-byte chip unique ID
TLV[ <del>TAG_6</del> TAG_6]	containing signature over the concatenated values of TLV[TAG_1], TLV[TAG_3], TLV[TAG_4] and TLV[TAG_5].

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The command is handled successfully.

**Return** The sm status.

**Parameters**

- [in] session\_ctx: The session context
- [in] inputData: The input data
- [in] inputDataLen: The input data length
- [in] attestationID: The attestation id
- [in] attestationAlgo: The attestation algorithm
- response: The response
- presponseLen: The presponse length
- ptimeStamp: The ptime stamp
- freshness: The freshness
- pfreshnessLen: The pfreshness length
- chipId: The chip identifier
- pchipIdLen: The pchip identifier length
- signature: The signature
- psignatureLen: The psignature length
- randomAttst: The random attst
- [in] randomAttstLen: The random attst length

## Function Se05x\_API\_ImportExternalObject

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

### Function Documentation

smStatus\_t **Se05x\_API\_ImportExternalObject** (pSe05xSession\_t *session\_ctx*, **const** uint8\_t \**ECKeydata*, size\_t *ECKeydataLen*, **const** uint8\_t \**ECAuthKeyID*, size\_t *ECAuthKeyIDLen*, **const** uint8\_t \**serializedObject*, size\_t *serializedObjectLen*)

Se05x\_API\_ImportExternalObject

Combined with the INS\_IMPORT\_EXTERNAL mask, enables users to send a WriteSecureObject APDU (WriteECKKey until WritePCR) protected by a secure channel.

Secure Objects can be imported into the SE05X through a secure channel which does not require the establishment of a session. This feature is also referred to single side import and can only be used to create or update objects.

The mechanism is based on ECKKey session to protect the Secure Object content and is summarized in the following figure.

External import flow

The flow above can be summarized in the following steps:

1. The user obtains the SE public key for import via the to get the public key from the device's key pair. Key ID 0x02 will return the public key of the EC key pair with RESERVED\_ID\_EXTERNAL\_IMPORT. The response is signed by the same key pair.
2. The user calls with input:
  - the applet AID (e.g.A0000003965453000000010300000000)
  - the SCPparameters
    - 1-byte SCP identifier, must equal 0xAB
    - 2-byte SCP parameter, must equal 0x01 followed by 1-byte security level (which follows the GlobalPlatform security level definition, see: .
  - key type, must be 0x88 (AES keytype)
  - key length, must be 0x10 (AES128key)
  - host public key (65-byte NIST P-256 publickey)
  - host public key curve identifier (must be 0x03 (=NIST\_P256))
  - ASN.1 signature over the TLV with tags 0xA6 and 0x7F49.

The applet will then calculate the master key by performing SHA256 over a byte array containing (in order):

- 4-byte counter value being 0x00000001
- shared secret (ECDH calculation according [IEEE P1363] using the private key from RESERVED\_ID\_ECKEY\_SESSION and the public key provided as input to ECKKeySessionInternalAuthenticate. The length depends on the curve used (e.g. 32 byte for NIST P-256 curve).
- 16-byte random generated by the SE05X.



- 2-byte SCP parameter, must equal 0x01 followed by 1-byte security level (which follows the GlobalPlatform security level definition, see: [. 1](#)).
- 1-byte keytype
- 1-byte keylength

The master key will then be the 16 MSB's of the hash output.

Using the master key, the 3 session keys are derived by following the GlobalPlatform specification to derive session keys, e.g. derivation input:

- ENCsession key = CMAC(MK, 000000000000000000000000400008001)
- CMACsession key = CMAC(MK, 000000000000000000000000600008001)
- RMACsession key = CMAC(MK, 000000000000000000000000700008001)

The Authentication Object ID needs to be passed using TAG\_IMPORT\_AUTH\_KEY\_ID, followed by the Write APDU command (using tag TAG\_1).

The Write APDU command needs to be constructed as follows:

- Encrypt the command encryption counter (starting with 0x00000000000000000000000000000001) using the S\_ENC key. This becomes the IV for the encrypted APDU.
- Get the APDU command payload and pad it (ISO9797 M2 padding).
- Encrypt the payload in AES CBC mode using the S\_ENC key.
- Set the Secure Messaging bit in the CLA (0x04).
- Concatenate the MAC chaining value with the full APDU.
- Then calculate the MAC on this byte array and append the 8-byte MAC value to the APDU.
- Finally increment the encryption counter for the next command.

A receipt will be generated by doing a CMAC operation on the input from tag 0xA6 and 0x7F49 using the RMAC session key,

$$\text{Receipt} = \text{CMAC}(\text{RMAC session key}, \langle \text{input from TLV 0xA6 and TLV 0x7F49} \rangle)$$

There is no need to establish a session; therefore, the `ImportExternalObject` commands are always sent in the default session. The `ImportExternalObject` commands are replayable.

The P1 and P2 parameters shall be coded as per the intended operation. For example, to import an EC Key, the P1 and P2 parameters as defined in WriteEckey shall be specified.

### Command to Applet

Field	Value	Description
CLA	0x80	
INS	INS_IMPORT_EXTERNAL	See <a href="#">SE05x_INS_t</a>
P1	P1_DEFAULT	See <a href="#">SE05x_P1_t</a>
P2	P2_DEFAULT	See <a href="#">SE05x_P2_t</a>
Lc	#(Payload)	
Pay-load	TLV[TAG_IMPORT_AUTH_DATA]	Authentication data
	TLV[TAG_IMPORT_AUTH_KEYID]	ECKey Public key Identifier
	TLV[TAG_1]...	Wraps a complete WriteSecureObject command, protected by ECKey session secure messaging
	TLV[TAG_11]	4-byte version [Optional]

*R-APDU Body*

NA

#### Parameters

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] ECKeydata: ECKeydata [1:kSE05x\_TAG\_2]
- [in] ECKeydataLen: Length of ECKeydata
- [in] serializedObject: serializedObject [2:kSE05x\_TAG\_3]
- [in] serializedObjectLen: Length of serializedObject

#### Function Se05x\_API\_ImportObject

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

#### Function Documentation

smStatus\_t **Se05x\_API\_ImportObject** (pSe05xSession\_t session\_ctx, uint32\_t objectID, [SE05x\\_RSAPublicKeyComponent\\_t](#) rsaKeyComp, const uint8\_t \*serializedObject, size\_t serializedObjectLen)

Se05x\_API\_ImportObject

Writes a serialized Secure Object to the SE05X (i.e., “import”)

*Command to Applet*

Field	Value	Description
P1	P1_DEFAULT	See <a href="#">SE05x_P1_t</a>
P2	P2_IMPORT	See <a href="#">SE05x_P2_t</a>
Pay-load	TLV[TAG_1]	4-byte identifier.
	TLV[TAG_2]	1-byte <a href="#">SE05x_RSAPublicKeyComponent_t</a> [Conditional: only when the identifier refers to an RSAKey object]
	TLV[TAG_3]	Serialized object (encrypted).

*R-APDU Body*

NA

*R-APDU Trailer*

#### Parameters

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] objectID: object id [1:kSE05x\_TAG\_1]
- [in] rsaKeyComp: rsaKeyComp [2:kSE05x\_TAG\_2]
- [in] serializedObject: serializedObject [3:kSE05x\_TAG\_3]
- [in] serializedObjectLen: Length of serializedObject

### Function Se05x\_API\_IncCounter

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

#### Function Documentation

smStatus\_t **Se05x\_API\_IncCounter** (pSe05xSession\_t session\_ctx, uint32\_t objectID)  
Se05x\_API\_IncCounter

See [Se05x\\_API\\_CreateCounter](#)

#### Parameters

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] objectID: object id [1:kSE05x\_TAG\_1]

### Function Se05x\_API\_MACFinal

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

#### Function Documentation

smStatus\_t **Se05x\_API\_MACFinal** (pSe05xSession\_t session\_ctx, const uint8\_t \*inputData, size\_t inputDataLen, SE05x\_CryptoObjectID\_t cryptoObjectID, const uint8\_t \*macValidateData, size\_t macValidateDataLen, uint8\_t \*macValue, size\_t \*pmacValueLen)

Se05x\_API\_MACFinal

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	See <a href="#">SE05x_INS_t</a>
P1	P1_MAC	See <a href="#">SE05x_P1_t</a>
P2	P2_FINAL	See <a href="#">SE05x_P2_t</a>
Pay-load	TLV[TAG_1]	Byte array containing data to be taken as input to MAC.
	TLV[TAG_2]	2-byte Crypto Object identifier
	TLV[TAG_3]	Byte array containing MAC to validate. [Conditional: only applicable the crypto object is set for validating (MACInit P2 = P2_VALIDATE)]
Le	0x00	Expecting MAC or result.

#### *R-APDU Body*

Value	Description
TLV[TAG_1]	MAC value (when MACInit had P2 = P2_GENERATE) or <a href="#">SE05x_Result_t</a> (when MACInit had P2 = P2_VERIFY).

#### *R-APDU Trailer*

SW	Description
SW_NO_ERROR	The command is handled successfully.

#### Parameters

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] inputData: inputData [1:kSE05x\_TAG\_1]
- [in] inputDataLen: Length of inputData
- [in] cryptoObjectID: cryptoObjectID [2:kSE05x\_TAG\_2]
- [in] macValidateData: macValidateData [3:kSE05x\_TAG\_3]
- [in] macValidateDataLen: Length of macValidateData
- [out] macValue: [0:kSE05x\_TAG\_1]
- [inout] pmacValueLen: Length for macValue

#### Function [Se05x\\_API\\_MACInit](#)

- Defined in file `hostlib_hostLib_se05x_03_xx_xx_se05x_APDU_apis.h`

## Function Documentation

smStatus\_t **Se05x\_API\_MACInit** (pSe05xSession\_t session\_ctx, uint32\_t objectID, SE05x\_CryptoObjectID\_t cryptoObjectID, const SE05x\_Mac\_Oper\_t mac\_oper)

Se05x\_API\_MACInit

Initiate a MAC operation. The state of the MAC operation is kept in the Crypto Object until it's finalized or deleted.

The 4-byte identifier of the key must refer to an AESKey, DESKey or HMACKey.

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	SE05x_INS_t
P1	P1_MAC	See SE05x_P1_t
P2	P2_GENERATE or P2_VALIDATE	See SE05x_P2_t
Lc	#(Payload)	
Payload	TLV[TAG_1]	4-byte identifier of the MAC key.
	TLV[TAG_2]	2-byte Crypto Object identifier
Le	0x00	

*R-APDU Body*

NA

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The command is handled successfully.

### Parameters

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] objectID: objectID [1:kSE05x\_TAG\_1]
- [in] cryptoObjectID: cryptoObjectID [2:kSE05x\_TAG\_2]
- [in] mac\_oper: The Operation

## Function Se05x\_API\_MACOneShot\_G

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

## Function Documentation

smStatus\_t **Se05x\_API\_MACOneShot\_G** (pSe05xSession\_t session\_ctx, uint32\_t objectID, uint8\_t macOperation, **const** uint8\_t \*inputData, size\_t inputDataLen, uint8\_t \*macValue, size\_t \*pmacValueLen)

Se05x\_API\_MACOneShot\_G

Generate. See *Se05x\_API\_MACOneShot\_V* for Verification.

Performs a MAC operation in one shot (without keeping state).

The 4-byte identifier of the key must refer to an AESKey, DESKey or HMACKey.

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_CRYPT0	<i>SE05x_INS_t</i>
P1	P1_MAC	See <i>SE05x_P1_t</i>
P2	P2_GENERATE_ONESHOT P2_VALIDATE_ONESHOT	or See <i>SE05x_P2_t</i>
Lc	#(Payload)	
Pay-load	TLV[TAG_1]	4-byte identifier of the key object.
	TLV[TAG_2]	1-byte MACAlgoRef
	TLV[TAG_3]	Byte array containing data to be taken as input to MAC.
	TLV[TAG_5]	MAC to verify (when P2=P2_VALIDATE_ONESHOT)
Le	0x00	Expecting MAC or Result.

*R-APDU Body*

Value	Description
TLV[TAG_1]	MAC value (P2=P2_GENERATE_ONESHOT) or <i>SE05x_Result_t</i> (when p2=P2_VALIDATE_ONESHOT).

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The command is handled successfully.

### Parameters

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] objectID: objectID [1:kSE05x\_TAG\_1]
- [in] macOperation: macOperation [2:kSE05x\_TAG\_2]
- [in] inputData: inputData [3:kSE05x\_TAG\_3]
- [in] inputDataLen: Length of inputData
- [out] macValue: [0:kSE05x\_TAG\_1]
- [inout] pmacValueLen: Length for macValue

## Function Se05x\_API\_MACOneShot\_V

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

### Function Documentation

smStatus\_t **Se05x\_API\_MACOneShot\_V** (pSe05xSession\_t *session\_ctx*, uint32\_t *objectID*, uint8\_t *macOperation*, **const** uint8\_t \**inputData*, size\_t *inputDataLen*, **const** uint8\_t \**MAC*, size\_t *MACLen*, uint8\_t \**macValue*, size\_t \**pmacValueLen*)

Se05x\_API\_MACOneShot\_V

Validate. See *Se05x\_API\_MACOneShot\_G* for Generation.

#### Parameters

- [in] *session\_ctx*: Session Context [0:kSE05x\_pSession]
- [in] *objectID*: objectID [1:kSE05x\_TAG\_1]
- [in] *macOperation*: macOperation [2:kSE05x\_TAG\_2]
- [in] *inputData*: inputData [3:kSE05x\_TAG\_3]
- [in] *inputDataLen*: Length of inputData
- [in] *MAC*: MAC to verify (when P2=P2\_VALIDATE\_ONESHOT) [4:kSE05x\_TAG\_5]
- [in] *MACLen*: Length of MAC
- [out] *macValue*: [0:kSE05x\_TAG\_1]
- [inout] *pmacValueLen*: Length for macValue

## Function Se05x\_API\_MACUpdate

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

### Function Documentation

smStatus\_t **Se05x\_API\_MACUpdate** (pSe05xSession\_t *session\_ctx*, **const** uint8\_t \**inputData*, size\_t *inputDataLen*, SE05x\_CryptoObjectID\_t *cryptoObjectID*)

Se05x\_API\_MACUpdate

Update MAC

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_CRYPT0	<i>SE05x_INS_t</i>
P1	P1_MAC	See <i>SE05x_P1_t</i>
P2	P2_UPDATE	See <i>SE05x_P2_t</i>
Lc	#(Payload)	
Payload	TLV[TAG_1]	Byte array containing data to be taken as input to MAC.
	TLV[TAG_2]	2-byte Crypto Object identifier
Le	•	

*R-APDU Body*

NA

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The command is handled successfully.

#### Parameters

- [in] *session\_ctx*: Session Context [0:kSE05x\_pSession]
- [in] *inputData*: inputData [1:kSE05x\_TAG\_1]
- [in] *inputDataLen*: Length of inputData
- [in] *cryptoObjectID*: cryptoObjectID [2:kSE05x\_TAG\_2]

#### Function Se05x\_API\_PBKDF2

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

#### Function Documentation

smStatus\_t **Se05x\_API\_PBKDF2** (pSe05xSession\_t *session\_ctx*, uint32\_t *objectID*, **const** uint8\_t \**salt*, size\_t *saltLen*, uint16\_t *count*, uint16\_t *requestedLen*, uint8\_t \**derivedSessionKey*, size\_t \**pderivedSessionKeyLen*)  
 Se05x\_API\_HKDF\_Extended

Only step 2 of the algorithm is executed, i.e. Expand only.

Using an IV as input parameter results in a FIPS compliant SP800-108 KDF in Feedback Mode where K[0] is the provided IV. This KDF is then using a 8-bit counter, AFTER\_FIXED counter location.

*Command to Applet*



Field	Value	Description
CLA	0x80	
INS	INS_CRYPT0	<a href="#">SE05x_INS_t</a>
P1	P1_DEFAULT	See <a href="#">SE05x_P1_t</a>
P2	P2_HKDF_EXPAND_ONLY	See <a href="#">SE05x_P2_t</a>
Lc	#(Payload)	
Payload	TLV[TAG_1]	4-byte HMACKey identifier (= PRK)
TLV[TAG_2]	2]-byte DigestMode (except DIGEST_NO_HASH)	
TLV[TAG_3]	3]-byte array (0-64 bytes) containing IV. [Optional] [Conditional: only when TLV[TAG_6] is absent.]	
TLV[TAG_4]	4]-byte info: The context and information to apply (1 to 80 bytes). [Optional]	
TLV[TAG_5]	5]-byte requested length (L): 1 up to MAX_APDU_PAYLOAD_LENGTH	
TLV[TAG_6]	6]-byte HMACKey identifier containing IV. [Optional] [Conditional: only when TLV[TAG_3] is absent]	
TLV[TAG_7]	7]-byte HMACKey identifier to store output. [Optional]	
Le	0x00	

*R-APDU Body*

Value	Description
TLV[TAG_1]	HKDF output. [Conditional: only when the input does not contain TLV[TAG-7]]

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The HKDF is executed successfully.

/

```
smStatus_t Se05x_API_HKDF_Extended(pSe05xSession_t session_ctx, uint32_t hmacID,
SE05x_DigestMode_t digestMode, SE05x_HkdfMode_t hkdfMode, const uint8_t salt, size_t saltLen,
uint32_t saltID, const uint8_t info, size_t infoLen, uint32_t derivedKeyID, uint16_t deriveDataLen,
uint8_t hkdfOuput, size_t phkdfOuputLen);
```

```
/* Se05x_API_PBKDF2
```

Password Based Key Derivation Function 2 (PBKDF2) according [RFC8018].

The password is an input to the KDF and must be stored inside the .

The output is returned to the host.

# Command to Applet

```
verbatim embed:rst:leading-asterisk +-----+-----+-----+-----+ |
Field | Value | Description | +-----+-----+-----+-----+ |
| CLA | 0x80 | | +-----+-----+-----+ | INS | INS_CRYPT0 |
| SE05x_INS_t | +-----+-----+-----+ | P1 | P1_DEFAULT |
| See SE05x_P1_t | +-----+-----+-----+ | P2 |
P2_PBKDF | See SE05x_P2_t | +-----+-----+-----+ |
+ | Lc | #(Payload) | | +-----+-----+-----+ | |
```

TLV[TAG\_1] | 4-byte password identifier (object type must be HMACKey) | +-----  
+-----+-----+ | TLV[TAG\_2] | Salt (0 to 64 bytes)  
[Optional] | +-----+-----+ | TLV[TAG\_3] | 2-byte  
Iteration count: 1 up to 0x7FFF. | +-----+-----+  
+ | TLV[TAG\_4] | 2-byte Requested length: 1 up to 512 bytes. | +-----  
+-----+-----+ | Le | 0x00 | Expecting derived key material. |  
+-----+-----+-----+-----+

*R-APDU Body*

Value	Description
TLV[TAG_1]	Derived key material (session key).

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The command is handled successfully.

### Parameters

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] objectID: 4-byte password identifier (object type must be HMACKey) [1:kSE05x\_TAG\_1]
- [in] salt: salt [2:kSE05x\_TAG\_2]
- [in] saltLen: Length of salt
- [in] count: count [3:kSE05x\_TAG\_3]
- [in] requestedLen: requestedLen [4:kSE05x\_TAG\_4]
- [out] derivedSessionKey: [0:kSE05x\_TAG\_1]
- [inout] pderivedSessionKeyLen: Length for derivedSessionKey

## Function Se05x\_API\_ReadCryptoObjectList

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

### Function Documentation

smStatus\_t **Se05x\_API\_ReadCryptoObjectList** (pSe05xSession\_t session\_ctx, uint8\_t \*idlist, size\_t \*pidlistLen)

Se05x\_API\_ReadCryptoObjectList

Get the list of allocated Crypto Objects indicating the identifier, the CryptoContext and the sub type of the CryptoContext.

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_READ	See <a href="#">SE05x_INS_t</a>
P1	P1_CRYPTOBJ	See <a href="#">SE05x_P1_t</a>
P2	P2_LIST	See <a href="#">SE05x_P2_t</a>
Le	0x00	

*R-APDU Body*

Value	Description
TLV[TAG_1]	Byte array containing a list of 2-byte Crypto Object identifiers, followed by 1-byte CryptoContext and 1-byte subtype for each Crypto Object (so 4 bytes for each Crypto Object).

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	Data is returned successfully.

**Parameters**

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [out] idlist: If more ids are present [0:kSE05x\_TAG\_1]
- [inout] pidlistLen: Length for idlist

**Function Se05x\_API\_ReadECCurveList**

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

**Function Documentation**

smStatus\_t **Se05x\_API\_ReadECCurveList** (pSe05xSession\_t session\_ctx, uint8\_t \*curveList, size\_t \*pcurveListLen)

Se05x\_API\_ReadECCurveList

Get a list of (Weierstrass) EC curves that are instantiated.

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_READ	See <a href="#">SE05x_INS_t</a>
P1	P1_CURVE	See <a href="#">SE05x_P1_t</a>
P2	P2_LIST	See <a href="#">SE05x_P2_t</a>
Le	0x00	

*R-APDU Body*

Value	Description
TLV[TAG_1]	Byte array listing all curve identifiers in <a href="#">SE05x_ECCurve_t</a> (excluding UNUSED) where the curve identifier < 0x40; for each curve, a 1-byte SetIndicatorRef is returned.

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	Data is returned successfully.

**Parameters**

- [in] `session_ctx`: Session Context [0:kSE05x\_pSession]
- [out] `curveList`: [0:kSE05x\_TAG\_1]
- [inout] `pcurveListLen`: Length for `curveList`

**Function Se05x\_API\_ReadIDList**

- Defined in file `hostlib_hostLib_se05x_03_xx_xx_se05x_APDU_apis.h`

**Function Documentation**

`smStatus_t Se05x_API_ReadIDList` (`pSe05xSession_t session_ctx`, `uint16_t outputOffset`, `uint8_t filter`, `uint8_t *pmore`, `uint8_t *idlist`, `size_t *pidlistLen`)

`Se05x_API_ReadIDList`

Get a list of present Secure Object identifiers.

The offset in TAG\_1 is an 0-based offset in the list of object. As the user does not know how many objects would be returned, the offset needs to be based on the return values from the previous `ReadIDList`. If the applet only returns a part of the result, it will indicate that more identifiers are available (by setting TLV[TAG\_1] in the response to 0x01). The user can then retrieve the next chunk of identifiers by calling `ReadIDList` with an offset that equals the amount of identifiers listed in the previous response.

*Example 1:* first `ReadIDList` command TAG\_1=0, response TAG\_1=0, TAG\_2=complete list

*Example 2:* first `ReadIDList` command TAG\_1=0, response TAG\_1=1, TAG\_2=first chunk (m entries) second `ReadIDList` command TAG\_1=m, response TAG\_1=1, TAG\_2=second chunk (n entries) third `ReadIDList` command TAG\_1=(m+n), response TAG\_1=0, TAG\_2=third last chunk

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_READ	See <a href="#">SE05x_INS_t</a>
P1	P1_DEFAULT	See <a href="#">SE05x_P1_t</a>
P2	P2_LIST	See <a href="#">SE05x_P2_t</a>
Lc	#(Payload)	
	TLV[TAG_1]	2-byte offset
	TLV[TAG_2]	1-byte type filter: 1 byte from <a href="#">SE05x_SecObjTyp_t</a> or 0xFF for all types.
Le	0x00	

*R-APDU Body*

Value	Description
TLV[TAG_1]	1-byte MoreIndicatorRef
TLV[TAG_2]	Byte array containing 4-byte identifiers.

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	Data is returned successfully.

**Parameters**

- [in] `session_ctx`: Session Context [0:kSE05x\_pSession]
- [in] `outputOffset`: output offset [1:kSE05x\_TAG\_1]
- [in] `filter`: filter [2:kSE05x\_TAG\_2]
- [out] `pmore`: If more ids are present [0:kSE05x\_TAG\_1]
- [out] `idlist`: Byte array containing 4-byte identifiers [1:kSE05x\_TAG\_2]
- [inout] `pidlistLen`: Length for idlist

**Function Se05x\_API\_ReadObject**

- Defined in file `_hostlib_hostLib_se05x_03_xx_xx_se05x_APDU_apis.h`

**Function Documentation**

`smStatus_t Se05x_API_ReadObject` (`pSe05xSession_t session_ctx`, `uint32_t objectID`, `uint16_t offset`, `uint16_t length`, `uint8_t *data`, `size_t *pdataLen`)

`Se05x_API_ReadObject`

Reads the content of a Secure Object.

- If the object is a key pair, the command will return the key pair's public key.
- If the object is a public key, the command will return the public key.
- If the object is a private key or a symmetric key or a userID, the command will return `SW_CONDITIONS_NOT_SATISFIED`.
- If the object is a binary file, the file content is read, giving the offset in `TLV[TAG_2]` and the length to read in `TLV[TAG_3]`. Both `TLV[TAG_2]` and `TLV[TAG_3]` are bound together; i.e.. either both tags are present, or both are absent. If both are absent, the whole file content is returned.
- If the object is a monotonic counter, the counter value is returned.
- If the object is a PCR, the PCR value is returned.
- If `TLV[TAG_4]` is filled, only the modulus or public exponent of an RSA key pair or RSA public key is read. It does not apply to other Secure Object types.

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_READ	See <code>SE05x_INS_t</code> , in addition to INS_READ, users can set the INS_ATTEST flag. In that case, attestation applies.
P1	P1_DEFAULT	See <code>SE05x_P1_t</code>
P2	P2_DEFAULT	See <code>SE05x_P2_t</code>
Lc	#(Pay-load)	Payload Length.
	TLV[TAG_1]	4byte object identifier
	TLV[TAG_2]	byte offset [Optional: default 0] [Conditional: only when the object is a BinaryFile object]
	TLV[TAG_3]	byte length [Optional: default 0] [Conditional: only when the object is a BinaryFile object]
	TLV[TAG_4]	byte <code>SE05x_RSAKeyComponent_t</code> : either RSA_COMP_MOD or RSA_COMP_PUB_EXP. [Optional] [Conditional: only for RSA key components]
Le	0x00	

#### *R-APDU Body*

Value	Description
TLV[TAG_1]	Data read from the secure object.

#### *R-APDU Trailer*

SW	Description
SW_NO_ERROR	The read is done successfully.

#### Parameters

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] objectID: object id [1:kSE05x\_TAG\_1]
- [in] offset: offset [2:kSE05x\_TAG\_2]
- [in] length: length [3:kSE05x\_TAG\_3]
- [out] data: [0:kSE05x\_TAG\_1]
- [inout] pdataLen: Length for data

#### Function `Se05x_API_ReadObject_W_Attest`

- Defined in file `hostlib_hostLib_se05x_03_xx_xx_se05x_APDU_apis.h`

## Function Documentation

`smStatus_t Se05x_API_ReadObject_W_Attest` (`pSe05xSession_t session_ctx`, `uint32_t objectID`, `uint16_t offset`, `uint16_t length`, `uint32_t attestID`, `SE05x_AttestationAlgo_t attestAlgo`, `const uint8_t *random`, `size_t randomLen`, `uint8_t *data`, `size_t *pdataLen`, `uint8_t *attribute`, `size_t *pattributeLen`, `SE05x_TimeStamp_t *ptimeStamp`, `uint8_t *outrandom`, `size_t *poutrandomLen`, `uint8_t *chipId`, `size_t *pchipIdLen`, `uint8_t *signature`, `size_t *psignatureLen`)

`Se05x_API_ReadObject_W_Attest`

Read with attestation.

See *Se05x\_API\_ReadObject*

When `INS_ATTEST` is set in addition to `INS_READ`, the secure object is read with attestation. In addition to the response in `TLV[TAG_1]`, there are additional tags:

`TLV[TAG_2]` will hold the object attributes (see `ObjectAttributes`).

`TLV[TAG_3]` relative timestamp when the object has been retrieved

`TLV[TAG_4]` will hold freshness random data

`TLV[TAG_5]` will hold the unique ID of the device.

`TLV[TAG_6]` will hold the signature over all concatenated Value fields tags of the response (`TAG_1` until and including `TAG_5`).

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_READ	See <i>SE05x_INS_t</i> , in addition to <code>INS_READ</code> , users can set the <code>INS_ATTEST</code> flag. In that case, attestation applies.
P1	P1_DEFAULT	See <i>SE05x_P1_t</i>
P2	P2_DEFAULT	See <i>SE05x_P2_t</i>
Lc	#{Pay-load}	Payload Length.
	<code>TLV[TAG_1]</code>	byte object identifier
	<code>TLV[TAG_2]</code>	byte offset [Optional: default 0] [Conditional: only when the object is a <code>BinaryFile</code> object]
	<code>TLV[TAG_3]</code>	byte length [Optional: default 0] [Conditional: only when the object is a <code>BinaryFile</code> object]
	<code>TLV[TAG_4]</code>	byte <i>SE05x_RSAPublicKeyComponent_t</i> : either <code>RSA_COMP_MOD</code> or <code>RSA_COMP_PUB_EXP</code> . [Optional] [Conditional: only for RSA key components]
	<code>TLV[TAG_5]</code>	byte attestation object identifier. [Optional] [Conditional: only when <code>INS_ATTEST</code> is set]
	<code>TLV[TAG_6]</code>	byte <i>SE05x_AttestationAlgo_t</i> [Optional] [Conditional: only when <code>INS_ATTEST</code> is set]
	<code>TLV[TAG_7]</code>	byte freshness random [Optional] [Conditional: only when <code>INS_ATTEST</code> is set]
Le	0x00	

Value	Description
TLV[TAG_1]	Data read from the secure object.
TLV[TAG_2]	only when INS_ATTEST is set) Byte array containing the attributes (see ObjectAttributesRef).
TLV[TAG_3]	only when INS_ATTEST is set) 12-byte timestamp
TLV[TAG_4]	only when INS_ATTEST is set) 16-byte freshness random
TLV[TAG_5]	only when INS_ATTEST is set) 18-byte Chip unique ID
TLV[TAG_6]	only when INS_ATTEST is set) Signature applied over the value of TLV[TAG_1], TLV[TAG_2], TLV[TAG_3], TLV[TAG_4] and TLV[TAG_5].

#### *R-APDU Body*

Value	Description
TLV[TAG_1]	Data read from the secure object.
TLV[TAG_2]	only when INS_ATTEST is set) Byte array containing the attributes (see ObjectAttributesRef).
TLV[TAG_3]	only when INS_ATTEST is set) 12-byte timestamp
TLV[TAG_4]	only when INS_ATTEST is set) 16-byte freshness random
TLV[TAG_5]	only when INS_ATTEST is set) 18-byte Chip unique ID
TLV[TAG_6]	only when INS_ATTEST is set) Signature applied over the value of TLV[TAG_1], TLV[TAG_2], TLV[TAG_3], TLV[TAG_4] and TLV[TAG_5].

**Return** The sm status.

#### **Parameters**

- [in] session\_ctx: The session context
- [in] objectID: The object id
- [in] offset: The offset
- [in] length: The length
- [in] attestID: The attest id
- [in] attestAlgo: The attest algorithm
- [in] random: The random
- [in] randomLen: The random length
- data: The data
- pdataLen: The pdata length
- attribute: The attribute
- pattributeLen: The pattribute length
- ptimeStamp: The ptime stamp
- outrandom: The outrandom
- poutrandomLen: The poutrandom length
- chipId: The chip identifier
- pchipIdLen: The pchip identifier length
- signature: The signature



- `psignatureLen`: The psignature length

## Function `Se05x_API_ReadObjectAttributes_W_Attest`

- Defined in `file_hostlib_hostLib_se05x_03_xx_xx_se05x_APDU_apis.h`

## Function Documentation

```
smStatus_t Se05x_API_ReadObjectAttributes_W_Attest (pSe05xSession_t session_ctx,
uint32_t objectID, uint32_t attestID,
SE05x_AttestationAlgo_t attestAlgo,
const uint8_t *random, size_t
randomLen, uint8_t *data, size_t
*pdataLen, SE05x_TimeStamp_t *pti-
meStamp, uint8_t *outrandom, size_t
*poutrandomLen, uint8_t *chipId, size_t
*pchipIdLen, uint8_t *signature, size_t
*psignatureLen)
```

`Se05x_API_ReadObjectAttributes_W_Attest`

Reads the attributes of a Secure Object (without the value of the Secure Object).

Each Secure Object has a number of attributes assigned to it. These attributes are listed in for Authentication Objects and in for non-Authentication Objects.

*Authentication Object attributes*

Attribute	Size (bytes)	Description
Object identifier	4	See <code>identifiersRef</code>
Object type	1	One of <code>SecureObjectType</code>
Authentication attribute	1	One of <code>SetIndicatorRef</code>
Object counter	2	Number of failed attempts for an authentication object if the Maximum Authentication Attempts has been set.
Authentication object identifier	4	”Owner” of the secure object; i.e., the identifier of the session authentication object when the object has been created.
Maximum authentication attempts	2	Maximum number of authentication attempts. 0 means unlimited.
Policy	Variable	Policy attached to the object
Origin	1	One of <code>OriginRef</code> ; indicates the origin of the Secure Object, either externally set, internally generated or trust provisioned by NXP.
Version	1	The Secure Object version. Default = 0. See FIPS compliance for details about versioning of Secure Objects.

*Non-Authentication Objects*

Attribute	Size (bytes)	Description
Object identifier	4	See Object identifiers
Object type	1	One of SecureObjectType
Authentication attribute	1	One of SetIndicatorRef
Tag length	2	Set to 0x0000, except for AESKey objects: for AESKey objects, this indicates the GMAC length that applies when doing AEAD operations. If the value is set to 0 and AEAD operations are done, the GMAC length shall be 128 bit.
Authentication object identifier	4	”Owner” of the secure object; i.e., the identifier of the session authentication object when the object has been created.
RFU	2	Set to 0x0000.
Policy	Variable	Policy attached to the object
Origin	1	One of OriginRef; indicates the origin of the Secure Object, either externally set, internally generated or trust provisioned by NXP.
Version	1	The Secure Object version. Default = 0. See FIPS compliance for details about versioning of Secure Objects.

#### Command to Applet

Field	Value	Description
CLA	0x80	
INS	INS_READ	See <a href="#">SE05x_INS_t</a> , in addition to INS_READ, users can set the INS_ATTEST flag. In that case, attestation applies.
P1	P1_DEFAULT	See <a href="#">SE05x_P1_t</a>
P2	P2_ATTRIBUTES	See <a href="#">SE05x_P2_t</a>
Lc	#(Payload)	Payload Length.
	TLV[TAG_1]	4-byte object identifier
	TLV[TAG_5]	4-byte attestation object identifier. [Optional] [Conditional: only when INS_ATTEST is set]
	TLV[TAG_6]	1-byte AttestationAlgo [Optional] [Conditional: only when INS_ATTEST is set]
	TLV[TAG_7]	16-byte freshness random [Optional] [Conditional: only when INS_ATTEST is set]
Le	0x00	

#### R-APDU Body

Value	Description
TLV[TAG_2]	2-byte array containing the attributes (see Object Attributes).
TLV[TAG_3]	(only when INS_ATTEST is set) 12-byte timestamp
TLV[TAG_4]	(only when INS_ATTEST is set) 16-byte freshness random
TLV[TAG_5]	(only when INS_ATTEST is set) 18-byte Chip unique ID
TLV[TAG_6]	(only when INS_ATTEST is set) Signature applied over the value of TLV[TAG_2], TLV[TAG_2], TLV[TAG_3], TLV[TAG_4] and TLV[TAG_5].

#### R-APDU Trailer

SW	Description
SW_NO_ERROR	The read is done successfully.

**Return** The sm status.

#### Parameters

- [in] `session_ctx`: The session context
- [in] `objectID`: The object id
- [in] `attestID`: The attest id
- [in] `attestAlgo`: The attest algorithm
- [in] `random`: The random
- [in] `randomLen`: The random length
- `data`: The data
- `pdataLen`: The pdata length
- `ptimeStamp`: The ptime stamp
- `outrandom`: The outrandom
- `poutrandomLen`: The poutrandom length
- `chipId`: The chip identifier
- `pchipIdLen`: The pchip identifier length
- `signature`: The signature
- `psignatureLen`: The psignature length

### Function `Se05x_API_ReadRSA`

- Defined in `file_hostlib_hostLib_se05x_03_xx_xx_se05x_APDU_apis.h`

### Function Documentation

`smStatus_t Se05x_API_ReadRSA` (`pSe05xSession_t session_ctx`, `uint32_t objectID`, `uint16_t offset`, `uint16_t length`, `SE05x_RSAPubKeyComp_t rsa_key_comp`, `uint8_t *data`, `size_t *pdataLen`)

`Se05x_API_ReadRSA`

See [Se05x\\_API\\_ReadObject](#)

#### Parameters

- [in] `session_ctx`: Session Context [0:kSE05x\_pSession]
- [in] `objectID`: object id [1:kSE05x\_TAG\_1]
- [in] `offset`: offset [2:kSE05x\_TAG\_2]
- [in] `length`: length [3:kSE05x\_TAG\_3]
- [in] `rsa_key_comp`: `rsa_key_comp` [4:kSE05x\_TAG\_4]
- [out] `data`: [0:kSE05x\_TAG\_1]

- [inout] pdataLen: Length for data

## Function Se05x\_API\_ReadRSA\_W\_Attest

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

## Function Documentation

smStatus\_t **Se05x\_API\_ReadRSA\_W\_Attest** (pSe05xSession\_t session\_ctx, uint32\_t objectID, uint16\_t offset, uint16\_t length, SE05x\_RSAPubKeyComp\_t rsa\_key\_comp, uint32\_t attestID, SE05x\_AttestationAlgo\_t attestAlgo, const uint8\_t \*random, size\_t randomLen, uint8\_t \*data, size\_t \*pdataLen, uint8\_t \*attribute, size\_t \*pattributeLen, SE05x\_TimeStamp\_t \*ptimeStamp, uint8\_t \*outrandom, size\_t \*poutrandomLen, uint8\_t \*chipId, size\_t \*pchipIdLen, uint8\_t \*signature, size\_t \*psignatureLen)

Se05x\_API\_ReadRSA\_W\_Attest

See [Se05x\\_API\\_ReadObject\\_W\\_Attest](#)

**Return** The sm status.

### Parameters

- [in] session\_ctx: The session context
- [in] objectID: The object id
- [in] offset: The offset
- [in] length: The length
- [in] rsa\_key\_comp: The rsa key component
- [in] attestID: The attest id
- [in] attestAlgo: The attest algorithm
- [in] random: The random
- [in] randomLen: The random length
- data: The data
- pdataLen: The pdata length
- attribute: The attribute
- pattributeLen: The pattribute length
- ptimeStamp: The ptime stamp
- outrandom: The outrandom
- poutrandomLen: The poutrandom length
- chipId: The chip identifier
- pchipIdLen: The pchip identifier length
- signature: The signature
- psignatureLen: The psignature length

## Function Se05x\_API\_ReadSize

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

## Function Documentation

smStatus\_t **Se05x\_API\_ReadSize** (pSe05xSession\_t *session\_ctx*, uint32\_t *objectID*, uint16\_t \**psize*)  
 Se05x\_API\_ReadSize

ReadSize

Get the size of a Secure Object (in bytes):

- For EC keys: the size of the curve is returned.
- For RSA keys: the key size is returned.
- For AES/DES/HMAC keys, the key size is returned.
- For binary files: the file size is returned
- For userIDs: nothing is returned (SW\_CONDITIONS\_NOT\_SATISFIED).
- For counters: the counter length is returned.
- For PCR: the PCR length is returned.

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_READ	See <a href="#">SE05x_INS_t</a>
P1	P1_DEFAULT	See <a href="#">SE05x_P1_t</a>
P2	P2_SIZE	See <a href="#">SE05x_P2_t</a>
Lc	#{Payload}	
	TLV[TAG_1]	4-byte object identifier.
Le	0x00	

*R-APDU Body*

Value	Description
TLV[TAG_1]	Byte array containing size.

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	Data is returned successfully.

**Return** The sm status.

### Parameters

- [in] *session\_ctx*: The session context
- [in] *objectID*: The object id
- *psize*: The psize

## Function Se05x\_API\_ReadType

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

### Function Documentation

smStatus\_t **Se05x\_API\_ReadType** (pSe05xSession\_t session\_ctx, uint32\_t objectID, SE05x\_SecureObjectType\_t \*ptype, uint8\_t \*pisTransient, const SE05x\_AttestationType\_t attestation\_type)

Se05x\_API\_ReadType

Get the type of a Secure Object.

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_READ	See <a href="#">SE05x_INS_t</a>
P1	P1_DEFAULT	See <a href="#">SE05x_P1_t</a>
P2	P2_TYPE	See <a href="#">SE05x_P2_t</a>
Lc	#{Payload}	
	TLV[TAG_1]	4-byte object identifier.
Lc	0x00	

*R-APDU Body*

Value	Description
TLV[TAG_1]	Type of the Secure Object: one of <a href="#">SE05x_SecObjTyp_t</a>
TLV[TAG_2]	TransientIndicatorRef

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	Data is returned successfully.

**Return** The sm status.

#### Parameters

- [in] session\_ctx: The session context
- [in] objectID: The object id
- ptype: The ptype
- pisTransient: The pis transient
- [in] attestation\_type: The attestation type

## Function Se05x\_API\_RefreshSession

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

### Function Documentation

smStatus\_t **Se05x\_API\_RefreshSession** (pSe05xSession\_t *session\_ctx*, pSe05xPolicy\_t *policy*)  
Se05x\_API\_RefreshSession

Refreshes a session on , the policy of the running session can be updated; the rest of the session state remains.

*Command to Applet*

Field	Value	Description
CLA	0x80	•
INS	INS_MGMT	See <a href="#">SE05x_INS_t</a>
P1	P1_DEFAULT	See <a href="#">SE05x_P1_t</a>
P2	P2_SESSION_REFRESH	See <a href="#">SE05x_P2_t</a>
Lc	#(Payload)	Payload length.
	TLV[TAG_POLICY]	Byte array containing the policy to attach to the session. [Optional]
Le	•	

*R-APDU Body*

NA

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The command is handled successfully.

### Parameters

- [in] *session\_ctx*: Session Context [0:kSE05x\_pSession]
- [in] *policy*: policy [1:kSE05x\_TAG\_POLICY]

## Function Se05x\_API\_RSADecrypt

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

## Function Documentation

smStatus\_t **Se05x\_API\_RSADecrypt** (pSe05xSession\_t session\_ctx, uint32\_t objectID, SE05x\_RSACryptionAlgo\_t rsaEncryptionAlgo, const uint8\_t \*inputData, size\_t inputDataLen, uint8\_t \*decryptedData, size\_t \*pdecryptedDataLen)

Se05x\_API\_RSADecrypt

The RSADecrypt command decrypts data.

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	SE05x_INS_t
P1	P1_RSA	See SE05x_P1_t
P2	P2_DECRYPT_ONESHOT	See SE05x_P2_t
Lc	#(Payload)	
Payload	TLV[TAG_1]	4-byte identifier of the key pair or private key.
	TLV[TAG_2]	1-byte SE05x_RSACryptionAlgo_t
	TLV[TAG_3]	Byte array containing data to be decrypted.
Le	0x00	Expected TLV with decrypted data.

*R-APDU Body*

Value	Description
TLV[TAG_1]	Encrypted data

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The command is handled successfully.

### Parameters

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] objectID: objectID [1:kSE05x\_TAG\_1]
- [in] rsaEncryptionAlgo: rsaEncryptionAlgo [2:kSE05x\_TAG\_2]
- [in] inputData: inputData [3:kSE05x\_TAG\_3]
- [in] inputDataLen: Length of inputData
- [out] decryptedData: [0:kSE05x\_TAG\_1]
- [inout] pdecryptedDataLen: Length for decryptedData



## Function Se05x\_API\_RSACrypt

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

## Function Documentation

smStatus\_t **Se05x\_API\_RSACrypt** (pSe05xSession\_t session\_ctx, uint32\_t objectID, SE05x\_RSAEncryptionAlgo\_t rsaEncryptionAlgo, const uint8\_t \*inputData, size\_t inputDataLen, uint8\_t \*encryptedData, size\_t \*pencryptedDataLen)

Se05x\_API\_RSACrypt

The RSACrypt command encrypts data.

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_CRYPT	SE05x_INS_t
P1	P1_RSA	See SE05x_P1_t
P2	P2_ENCRYPT_ONESHOT	See SE05x_P2_t
Lc	#(Payload)	
Payload	TLV[TAG_1]	4-byte identifier of the key pair or public key.
	TLV[TAG_2]	1-byte SE05x_RSAEncryptionAlgo_t
	TLV[TAG_3]	Byte array containing data to be encrypted.
Le	0x00	Expected TLV with encrypted data.

*R-APDU Body*

Value	Description
TLV[TAG_1]	Encrypted data

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The command is handled successfully.

## Parameters

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] objectID: objectID [1:kSE05x\_TAG\_1]
- [in] rsaEncryptionAlgo: rsaEncryptionAlgo [2:kSE05x\_TAG\_2]
- [in] inputData: inputData [3:kSE05x\_TAG\_3]
- [in] inputDataLen: Length of inputData
- [out] encryptedData: [0:kSE05x\_TAG\_1]
- [inout] pencryptedDataLen: Length for encryptedData

## Function Se05x\_API\_RSASign

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

## Function Documentation

smStatus\_t **Se05x\_API\_RSASign** (pSe05xSession\_t session\_ctx, uint32\_t objectID, SE05x\_RSASignatureAlgo\_t rsaSigningAlgo, const uint8\_t \*inputData, size\_t inputDataLen, uint8\_t \*signature, size\_t \*psignatureLen)

Se05x\_API\_RSASign

The RSASign command signs the input message using an RSA private key.

Name	Value	Description
RSA_SHA1_PKCS1_PSS	0x15	RFC8017: RSASSA-PSS
RSA_SHA224_PKCS1_PSS	0x2B	RFC8017: RSASSA-PSS
RSA_SHA256_PKCS1_PSS	0x2C	RFC8017: RSASSA-PSS
RSA_SHA384_PKCS1_PSS	0x2D	RFC8017: RSASSA-PSS
RSA_SHA512_PKCS1_PSS	0x2E	RFC8017: RSASSA-PSS
RSA_SHA1_PKCS1	0x0A	RFC8017: RSASSA-PKCS1-v1_5
RSA_SHA_224_PKCS1	0x27	RFC8017: RSASSA-PKCS1-v1_5
RSA_SHA_256_PKCS1	0x28	RFC8017: RSASSA-PKCS1-v1_5
RSA_SHA_384_PKCS1	0x29	RFC8017: RSASSA-PKCS1-v1_5
RSA_SHA_512_PKCS1	0x2A	RFC8017: RSASSA-PKCS1-v1_5

## Command to Applet

Field	Value	Description
CLA	0x80	
INS	INS_CRYPT0	SE05x_INS_t
P1	P1_SIGNATURE	See SE05x_P1_t
P2	P2_SIGN	See SE05x_P2_t
Lc	#(Payload)	
	TLV[TAG_1]	4-byte identifier of the key pair or private key.
	TLV[TAG_2]	1-byte SE05x_RSASignAlgo_t
	TLV[TAG_3]	Byte array containing input data.
Le	0x00	Expecting ASN.1 signature.

## R-APDU Body

Value	Description
TLV[TAG_1]	RSA signature in ASN.1 format.

## R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

## Parameters

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]

- [in] objectID: objectID [1:kSE05x\_TAG\_1]
- [in] rsaSigningAlgo: rsaSigningAlgo [2:kSE05x\_TAG\_2]
- [in] inputData: inputData [3:kSE05x\_TAG\_3]
- [in] inputDataLen: Length of inputData
- [out] signature: [0:kSE05x\_TAG\_1]
- [inout] psignatureLen: Length for signature

### Function Se05x\_API\_RSASVerify

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

### Function Documentation

smStatus\_t **Se05x\_API\_RSASVerify** (pSe05xSession\_t session\_ctx, uint32\_t objectID, SE05x\_RSASignatureAlgo\_t rsaSigningAlgo, const uint8\_t \*inputData, size\_t inputDataLen, const uint8\_t \*signature, size\_t signatureLen, SE05x\_Result\_t \*presult)

Se05x\_API\_RSASVerify

The RSASVerify command verifies the given signature and returns the result.

The key cannot be passed externally to the command directly. In case users want to use the command to verify signatures using different public keys or the public key value regularly changes, the user should create a transient key object to which the key value is written and then the identifier of that transient secure object can be used by this RSASVerify command.

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_CRYPT0	SE05x_INS_t
P1	P1_SIGNATURE	See SE05x_P1_t
P2	P2_VERIFY	See SE05x_P2_t
Lc	#(Payload)	
Payload		
	TLV[TAG_1]	4-byte identifier of the key pair or public key.
	TLV[TAG_2]	1-byte SE05x_RSASignAlgo_t
	TLV[TAG_3]	Byte array containing data to be verified.
	TLV[TAG_5]	Byte array containing ASN.1 signature.
Le	0x03	Expecting Result in TLV

*R-APDU Body*

Value	Description
TLV[TAG_1]	SE05x_Result_t: Verification result

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The command is handled successfully.

**Parameters**

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] objectID: objectID [1:kSE05x\_TAG\_1]
- [in] rsaSigningAlgo: rsaSigningAlgo [2:kSE05x\_TAG\_2]
- [in] inputData: inputData [3:kSE05x\_TAG\_3]
- [in] inputDataLen: Length of inputData
- [in] signature: signature [4:kSE05x\_TAG\_5]
- [in] signatureLen: Length of signature
- [out] presult: [0:kSE05x\_TAG\_1]

**Function Se05x\_API\_SetAppletFeatures**

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

**Function Documentation**

smStatus\_t **Se05x\_API\_SetAppletFeatures** (pSe05xSession\_t session\_ctx, [SE05x\\_Variant\\_t](#) variant)  
Se05x\_API\_SetAppletFeatures

Sets the applet features that are supported. To successfully execute this command, the session must be authenticated using the RESERVED\_ID\_FEATURE.

The 2-byte input value is a pre-defined AppletConfig value.

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See <a href="#">SE05x_INS_t</a>
P1	P1_DEFAULT	See <a href="#">SE05x_P1_t</a>
P2	P2_VARIANT	See <a href="#">SE05x_P2_t</a>
Lc	#(Payload)	Payload length
Payload	TLV[TAG_1]	2-byte Variant from <a href="#">SE05x_AppletConfig_t</a>

*R-APDU Body*

NA

*R-APDU Trailer*

**Parameters**

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] variant: variant [1:kSE05x\_TAG\_1]

## Function Se05x\_API\_SetCounterValue

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

## Function Documentation

smStatus\_t **Se05x\_API\_SetCounterValue** (pSe05xSession\_t session\_ctx, uint32\_t objectID, uint16\_t size, uint64\_t value)

Se05x\_API\_SetCounterValue

See *Se05x\_API\_CreateCounter*

### Parameters

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] objectID: object id [1:kSE05x\_TAG\_1]
- [in] size: size [3:kSE05x\_TAG\_2]
- [in] value: value [4:kSE05x\_TAG\_3]

## Function Se05x\_API\_SetECCurveParam

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

## Function Documentation

smStatus\_t **Se05x\_API\_SetECCurveParam** (pSe05xSession\_t session\_ctx, *SE05x\_ECCurve\_t* curveID, *SE05x\_ECCurveParam\_t* ecCurveParam, const uint8\_t \*inputData, size\_t inputDataLen)

Se05x\_API\_SetECCurveParam

Set a curve parameter. The curve must have been created first by CreateEcCurve.

All parameters must match the expected value for the listed curves. If the curve parameters are not correct, the curve cannot be used.

Users have to set all 5 curve parameters for the curve to be usable. Once all curve parameters are given, the secure element will check if all parameters are correct and return SW\_NO\_ERROR..

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_WRITE	See <i>SE05x_INS_t</i>
P1	P1_CURVE	See <i>SE05x_P1_t</i>
P2	P2_PARAM	See <i>SE05x_P2_t</i>
Lc	#(Payload)	
	TLV[TAG_1]	1-byte curve identifier, from <i>SE05x_ECCurve_t</i>
	TLV[TAG_2]	1-byte <i>SE05x_ECCurveParam_t</i>
	TLV[TAG_3]	Bytestring containing curve parameter value.

*R-APDU Body*

NA

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	Data is returned successfully.

#### Parameters

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] curveID: curve id [1:kSE05x\_TAG\_1]
- [in] ecCurveParam: ecCurveParam [2:kSE05x\_TAG\_2]
- [in] inputData: inputData [3:kSE05x\_TAG\_3]
- [in] inputDataLen: Length of inputData

### Function Se05x\_API\_SetLockState

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

#### Function Documentation

smStatus\_t **Se05x\_API\_SetLockState** (pSe05xSession\_t session\_ctx, uint8\_t lockIndicator, uint8\_t lockState)

Se05x\_API\_SetLockState

Sets the applet transport lock (locked or unlocked). There is a Persistent lock and a Transient Lock. If the Persistent lock is UNLOCKED, the device is unlocked (regardless of the Transient lock). If the Persistent lock is LOCKED, the device is only unlocked when the Transient lock is UNLOCKED and the device will be locked again after deselect of the applet.

Note that regardless of the lock state, the credential RESERVED\_ID\_TRANSPORT allows access to all features. For example, it is possible to write/update objects within the session opened by RESERVED\_ID\_TRANSPORT, even if the applet is locked.

The default TRANSIENT\_LOCK state is LOCKED; there is no default PERSISTENT\_LOCK state (depends on product configuration).

This command can only be used in a session that used the credential with identifier RESERVED\_ID\_TRANSPORT as authentication object.

PERSIS- TENT_LOCK	TRAN- SIENT_LOCK	Behavior
UNLOCKED	UNLOCKED	Unlocked until PERSISTENT_LOCK set to LOCKED.
UNLOCKED	LOCKED	Unlocked until PERSISTENT_LOCK set to LOCKED.
LOCKED	UNLOCKED	Unlocked until deselect or TRANSIENT_LOCK set to LOCKED.
LOCKED	LOCKED	Locked until PERSISTENT_LOCK set to UNLOCKED.

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See <i>SE05x_INS_t</i>
P1	P1_DEFAULT	See <i>SE05x_P1_t</i>
P2	P2_TRANSPORT	See <i>SE05x_P2_t</i>
Lc	#{Payload}	
Payload	TLV[TAG_1]	1-byte LockIndicatorRef
	TLV[TAG_2]	1-byte LockStateRef
Le		

*R-APDU Body*

NA

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The command is handled successfully.

#### Parameters

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] lockIndicator: lock indicator [1:kSE05x\_TAG\_1]
- [in] lockState: lock state [2:kSE05x\_TAG\_2]

#### Function Se05x\_API\_SetPlatformSCPRequest

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

#### Function Documentation

smStatus\_t **Se05x\_API\_SetPlatformSCPRequest** (pSe05xSession\_t *session\_ctx*,  
*SE05x\_PlatformSCPRequest\_t* *platformSCPRe-*  
*quest*)

Se05x\_API\_SetPlatformSCPRequest

Sets the required state for platform SCP (required or not required). This is a persistent state.

If platform SCP is set to SCP\_REQUIRED, any applet APDU command will be refused by the applet when platform SCP is not enabled. Enabled means full encryption and MAC, both on C-APDU and R-APDU. Any other level is not sufficient and will not be accepted. SCP02 will not be accepted (as there is no response MAC and encryption).

If platform SCP is set to “not required,” any applet APDU command will be accepted by the applet.

This command can only be used in a session that used the credential with identifier RE-SERVED\_ID\_PLATFORM\_SCP as authentication object.

Note that the default state is SCP\_NOT\_REQUIRED.

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See <i>SE05x_INS_t</i>
P1	P1_DEFAULT	See <i>SE05x_P1_t</i>
P2	P2_SCP	See <i>SE05x_P2_t</i>
Lc	#(Payload)	
Payload	TLV[TAG_1]	1-byte <i>SE05x_PlatformSCPRequest_t</i>
Le		

*R-APDU Body*

NA

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The command is handled successfully.

#### Parameters

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] platformSCPRequest: platf scp req [1:kSE05x\_TAG\_1]

#### Function Se05x\_API\_TLSCalculatePreMasterSecret

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

#### Function Documentation

smStatus\_t **Se05x\_API\_TLSCalculatePreMasterSecret** (pSe05xSession\_t session\_ctx, uint32\_t keyPairId, uint32\_t pskId, uint32\_t hmacKeyId, **const** uint8\_t \*inputData, size\_t inputDataLen)

Se05x\_API\_TLSCalculatePreMasterSecret

The command TLSCalculatePreMasterSecret will compute the pre-master secret for TLS according [RFC5246]. The pre-master secret will always be stored in an HMACKey object (TLV[TAG\_3]). The HMACKey object must be created before; otherwise the calculation of the pre-master secret will fail.

It can use one of these algorithms: - - - -

- PSK Key Exchange algorithm as defined in [RFC4279]
- RSA\_PSK Key Exchange algorithm as defined in [RFC4279]
- ECDHE\_PSK Key Exchange algorithm as defined in [RFC5489]
- EC Key Exchange algorithm as defined in [RFC4492]
- RSA Key Exchange algorithm as defined in [RFC5246]

TLV[TAG\_1] needs to be an (existing) HMACKey identifier containing the pre- shared Key.

Input data in TLV[TAG\_4] are:



- An EC public key when TLV[TAG\_2] refers to an EC key pair.
- An RSA encrypted secret when TLV[TAG\_2] refers to an RSA key pair.
- Empty when TLV[TAG\_2] is absent or empty.

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	See <i>SE05x_INS_t</i>
P1	P1_TLS	See <i>SE05x_P1_t</i>
P2	P2_PMS	See <i>SE05x_P2_t</i>
Lc	#(Payload)	
	TLV[TAG_1]	4-byte PSK identifier referring to a 16, 32, 48 or 64-byte Pre Shared Key. [Optional]
	TLV[TAG_2]	4-byte key pair identifier. [Optional]
	TLV[TAG_3]	4-byte target HMACKey identifier.
	TLV[TAG_4]	Byte array containing input data.
Le	•	

*R-APDU Body*

NA

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The command is handled successfully.

**Parameters**

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] keyPairId: keyPairId [1:kSE05x\_TAG\_1]
- [in] pskId: pskId [2:kSE05x\_TAG\_2]
- [in] hmacKeyId: hmacKeyId [3:kSE05x\_TAG\_3]
- [in] inputData: inputData [4:kSE05x\_TAG\_4]
- [in] inputDataLen: Length of inputData

## Function Se05x\_API\_TLSGenerateRandom

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

### Function Documentation

smStatus\_t **Se05x\_API\_TLSGenerateRandom** (pSe05xSession\_t session\_ctx, uint8\_t \*randomValue, size\_t \*prandomValueLen)

Se05x\_API\_TLSGenerateRandom

Generates a random that is stored in the SE05X and used by TLSPerformPRF.

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_CRYPT0	See <a href="#">SE05x_INS_t</a>
P1	P1_TLS	See <a href="#">SE05x_P1_t</a>
P2	P2_RANDOM	See <a href="#">SE05x_P2_t</a>
Lc	#(Payload)	
Le	0x22	Expecting TLV with 32 bytes data.

*R-APDU Body*

Value	Description
TLV[TAG_1]	32-byte random value

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The command is handled successfully.

### Parameters

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [out] randomValue: [0:kSE05x\_TAG\_1]
- [inout] prandomValueLen: Length for randomValue

## Function Se05x\_API\_TLSPerformPRF

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

## Function Documentation

smStatus\_t **Se05x\_API\_TLSPPerformPRF** (pSe05xSession\_t *session\_ctx*, uint32\_t *objectID*, uint8\_t *digestAlgo*, **const** uint8\_t *\*label*, size\_t *labelLen*, **const** uint8\_t *\*random*, size\_t *randomLen*, uint16\_t *reqLen*, uint8\_t *\*outputData*, size\_t *\*poutputDataLen*, **const** *SE05x\_TLSPPerformPRFType\_t* *tlsprf*)

Se05x\_API\_TLSPPerformPRF

The command TLSPPerformPRF will compute either:

- the master secret for TLS according [RFC5246], section 8.1
- key expansion data from a master secret for TLS according [RFC5246], section 6.3

Each time before calling this function, TLSGenerateRandom must be called. Executing this function will clear the random that is stored in the SE05X .

The function can be called as client or as server and either using the pre- master secret or master secret as input, stored in an HMACKey. The input length must be either 16, 32, 48 or 64 bytes.

This results in P2 having 4 possibilities:

- P2\_TLS\_PRF\_CLI\_HELLO: pass the clientHelloRandom to calculate a master secret, the serverHelloRandom is in SE05X , generated by TLSGenerateRandom.
- P2\_TLS\_PRF\_SRV\_HELLO: pass the serverHelloRandom to calculate a master secret, the clientHelloRandom is in SE05X , generated by TLSGenerateRandom.
- P2\_TLS\_PRF\_CLI\_RANDOM: pass the clientRandom to generate key expansion data, the serverRandom is in SE05X , generated by TLSGenerateRandom.
- P2\_TLS\_PRF\_SRV\_RANDOM: pass the serverRandom to generate key expansion data, the clientRandom is in SE05X

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	See <i>SE05x_INS_t</i>
P1	P1_TLS	See <i>SE05x_P1_t</i>
P2	See description above.	See <i>SE05x_P2_t</i>
Lc	#(Payload)	
	TLV[TAG_1]	4-byte HMACKey identifier.
	TLV[TAG_2]	1-byte <i>SE05x_DigestMode_t</i> , except DIGEST_NO_HASH.
	TLV[TAG_3]	Label (1 to 64 bytes)
	TLV[TAG_4]	32-byte random
	TLV[TAG_5]	2-byte requested length
Le	0x00	

*R-APDU Body*

Value	Description
TLV[TAG_1]	Byte array containing requested output data.

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The command is handled successfully.

**Return** The sm status.

**Parameters**

- [in] session\_ctx: The session context
- [in] objectID: The object id
- [in] digestAlgo: The digest algorithm
- [in] label: The label
- [in] labelLen: The label length
- [in] random: The random
- [in] randomLen: The random length
- [in] reqLen: The request length
- outputData: The output data
- poutputDataLen: The poutput data length
- [in] tlsprf: The tlsprf

## Function Se05x\_API\_VerifySessionUserID

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

## Function Documentation

smStatus\_t **Se05x\_API\_VerifySessionUserID** (pSe05xSession\_t session\_ctx, **const** uint8\_t \*userId, size\_t userIdLen)

Se05x\_API\_VerifySessionUserID

Verifies the session user identifier (UserID) in order to allow setting up a session. If the UserID is correct, the session establishment is successful; otherwise the session cannot be opened (SW\_CONDITIONS\_NOT\_SATISFIED is returned).

*Command to Applet*

Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See <a href="#">SE05x_INS_t</a>
P1	P1_DEFAULT	See <a href="#">SE05x_P1_t</a>
P2	P2_SESSION_USERID	See <a href="#">SE05x_P2_t</a>
Lc	#(Payload)	Payload length.
	TLV[TAG_1]	UserID value
Le	•	

*R-APDU Body*

NA

*R-APDU Trailer*

SW	Description
SW_NO_ERROR	The command is handled successfully.

**Parameters**

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] userId: userId [1:kSE05x\_TAG\_1]
- [in] userIdLen: Length of userId

**Function Se05x\_API\_WriteBinary**

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

**Function Documentation**

smStatus\_t **Se05x\_API\_WriteBinary** (pSe05xSession\_t session\_ctx, pSe05xPolicy\_t policy, uint32\_t objectID, uint16\_t offset, uint16\_t length, **const** uint8\_t \*inputData, size\_t inputDataLen)

Se05x\_API\_WriteBinary

Creates or writes to a binary file object. Data are written to either the start of the file or (if specified) to the offset passed to the function.

*Command to Applet*

Field	Value	Description
P1	P1_BINARY	See <a href="#">SE05x_P1_t</a>
P2	P2_DEFAULT	See <a href="#">SE05x_P2_t</a>
Pay-load	TLV[TAG_POLICY]	4-byte array containing the object policy. [Optional: default policy applies] [Conditional: only when the object identifier is not in use yet]
	TLV[TAG_1]	4-byte object identifier
	TLV[TAG_2]	2-byte file offset [Optional: default = 0]
	TLV[TAG_3]	2-byte file length (up to 0x7FFF). [Conditional: only when the object identifier is not in use yet]
	TLV[TAG_4]	Data to be written [Optional: if not given, TAG_3 must be filled]
	TLV[TAG_11]	4-byte version [Optional]

**Parameters**

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] policy: policy [1:kSE05x\_TAG\_POLICY]
- [in] objectID: object id [2:kSE05x\_TAG\_1]
- [in] offset: offset [3:kSE05x\_TAG\_2]
- [in] length: length [4:kSE05x\_TAG\_3]
- [in] inputData: input data [5:kSE05x\_TAG\_4]

- [in] inputDataLen: Length of inputData

## Function Se05x\_API\_WriteEckKey

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

## Function Documentation

```
smStatus_t Se05x_API_WriteEckKey (pSe05xSession_t session_ctx, pSe05xPolicy_t policy,
SE05x_MaxAttempts_t maxAttempt, uint32_t objectID,
SE05x_ECCurve_t curveID, const uint8_t *privKey, size_t
privKeyLen, const uint8_t *pubKey, size_t pubKeyLen, const
SE05x_INS_t ins_type, const SE05x_KeyPart_t key_part)
```

Se05x\_API\_WriteEckKey

Write or update an EC key object.

P1KeyPart indicates the key type to be created (if the object does not yet exist).

If P1KeyPart = P1\_KEY\_PAIR, Private Key Value (TLV[TAG\_3]) and Public Key Value (TLV[TAG\_4]) must both be present, or both be absent. If absent, the key pair is generated in the SE05X .

If the object already exists, P1KeyPart is ignored.

Field	Value	Description
P1	SE05x_P1_t   P1_EC	See SE05x_P1_t , P1KeyType should only be set for new objects.
P2	P2_DEFAULT	See P2
Pay-load	TLV[TAG_POLICY]	Object array containing the object policy. [Optional: default policy applies] [Conditional - only when the object identifier is not in use yet]
	TLV[TAG_MAXATTEMPTS]	MAXATTEMPTS number of attempts. If 0 is given, this means unlimited. [Optional: default unlimited] [Conditional: only when the object identifier is not in use yet and INS includes INS_AUTH_OBJECT; see AuthenticationObjectPolicies ]
	TLV[TAG_1]	4-byte object identifier
	TLV[TAG_2]	1-byte curve identifier, see ECCurve [Conditional: only when the object identifier is not in use yet; ]
	TLV[TAG_3]	Private key value (see ECKKeyRef ) [Conditional: only when the private key is externally generated and P1KeyType is either P1_KEY_PAIR or P1_PRIVATE]
	TLV[TAG_4]	Public key value (see ECKKeyRef ) [Conditional: only when the public key is externally generated and P1KeyType is either P1_KEY_PAIR or P1_PUBLIC]
	TLV[TAG_11]	4-byte version [Optional]

**Return** The sm status.

### Parameters

- [in] session\_ctx: The session context
- [in] policy: The policy
- [in] maxAttempt: The maximum attempt
- [in] objectID: The object id
- [in] curveID: The curve id
- [in] privKey: The priv key

- [in] privKeyLen: The priv key length
- [in] pubKey: The pub key
- [in] pubKeyLen: The pub key length
- [in] ins\_type: The insert type
- [in] key\_part: The key part

### Function Se05x\_API\_WritePCR

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

### Function Documentation

smStatus\_t **Se05x\_API\_WritePCR** (pSe05xSession\_t *session\_ctx*, pSe05xPolicy\_t *policy*, uint32\_t *pcrID*, **const** uint8\_t *\*initialValue*, size\_t *initialValueLen*, **const** uint8\_t *\*inputData*, size\_t *inputDataLen*)

Se05x\_API\_WritePCR

Creates or writes to a PCR object.

A PCR is a hash to which data can be appended; i.e., writing data to a PCR will update the value of the PCR to be the hash of all previously inserted data concatenated with the new input data.

A PCR will always use DigestMode = DIGEST\_SHA256; no other configuration possible.

If TAG\_2 and TAG\_3 is not passed, the PCR is reset to its initial value (i.e., the value set when the PCR was created).

This reset is controlled under the POLICY\_OBJ\_ALLOW\_DELETE policy, so users that can delete the PCR can also reset the PCR to initial value.

*Command to Applet*

Field	Value	Description
P1	P1_PCR	See <a href="#">SE05x_P1_t</a>
P2	P2_DEFAULT	See <a href="#">SE05x_P2_t</a>
Pay-load	TLV[TAG_POLICY]	Policy array containing the object policy. [Optional: default policy applies] [Conditional: only when the object identifier is not in use yet]
	TLV[TAG_1]	4-byte PCR identifier.
	TLV[TAG_2]	Initial hash value [Conditional: only when the object identifier is not in use yet]
	TLV[TAG_3]	Data to be extended to the existing PCR. [Conditional: only when the object identifier is already in use] [Optional: not present if a Reset is requested]

*R-APDU Body*

NA

*R-APDU Trailer*

### Parameters

- [in] session\_ctx: Session Context [0:kSE05x\_pSession]
- [in] policy: policy [1:kSE05x\_TAG\_POLICY]
- [in] pcrID: object id [2:kSE05x\_TAG\_1]

- [in] `initialValue`: `initialValue [3:kSE05x_TAG_2]`
- [in] `initialValueLen`: Length of `initialValue`
- [in] `inputData`: `inputData [4:kSE05x_TAG_3]`
- [in] `inputDataLen`: Length of `inputData`

## Function `Se05x_API_WriteRSAKey`

- Defined in file `_hostlib_hostLib_se05x_03_xx_xx_se05x_APDU_apis.h`

## Function Documentation

`smStatus_t Se05x_API_WriteRSAKey` (`pSe05xSession_t session_ctx`, `pSe05xPolicy_t policy`, `uint32_t objectID`, `uint16_t size`, `const uint8_t *p`, `size_t pLen`, `const uint8_t *q`, `size_t qLen`, `const uint8_t *dp`, `size_t dpLen`, `const uint8_t *dq`, `size_t dqLen`, `const uint8_t *qInv`, `size_t qInvLen`, `const uint8_t *pubExp`, `size_t pubExpLen`, `const uint8_t *priv`, `size_t privLen`, `const uint8_t *pubMod`, `size_t pubModLen`, `const SE05x_INS_t transient_type`, `const SE05x_KeyPart_t key_part`, `const SE05x_RSAPKeyFormat_t rsa_format`)

`Se05x_API_WriteRSAKey`

Creates or writes an RSA key or a key component.

Supported key sizes are listed in `RSABitLength`. Other values are not supported.

An RSA key creation requires multiple ADPUs to be sent:

- The first APDU must contain:
  - Policy (optional, so only if non-default applies)
  - Object identifier
  - Key size
  - 1 of the key components.
- Each next APDU must contain 1 of the key components.

The policy applies only once all key components are set.

Once an `RSAPKey` object has been created, its format remains fixed and cannot be updated (so CRT or raw mode, no switch possible).

If the object already exists, `P1KeyType` is ignored.

For key pairs, if no component is present (`TAG_3` until `TAG_9`), the key pair will be generated on chip; otherwise the key pair will be constructed starting with the given component.

For private keys or public keys, there should always be exactly one of the tags `TAG_3` until `TAG_10`.

- `TLV[TAG_8]` and `TLV[TAG_10]` must only contain a value if the key pair is to be set to a known value and `P1KeyType` is either `P1_KEY_PAIR` or `P1_PUBLIC`; otherwise the value must be absent and the length must be equal to 0.



- TLV[TAG\_9] must only contain a value if the key is to be set in raw mode to a known value and P1KeyType is either P1\_KEY\_PAIR or P1\_PRIVATE; otherwise the value must be absent and the length must be equal to 0.
- If TLV[TAG\_3] up to TLV[TAG\_10] are absent (except TLV[TAG\_8]), the RSA key will be generated on chip in case the object does not yet exist; otherwise it will be regenerated. This only applies to RSA key pairs.
- Keys can be set by setting the different components of a key; only 1 component can be set at a time in this case.

Field	Value	Description
P1	<a href="#">SE05x_KeyPart_t</a>   P1_RSA	See <a href="#">SE05x_P1_t</a>
P2	P2_DEFAULT or P2_RAW	See <a href="#">SE05x_P2_t</a> ; P2_RAW only in case P1KeyPart = P1_KEY_PAIR and TLV[TAG_3] until TLV[TAG_10] is empty and the must generate a raw RSA key pair; all other cases: P2_DEFAULT.
Pay-load	TLV[TAG_POLICY]	Byte array containing the object policy. [Optional: default policy applies] [Conditional: only when the object identifier is not in use yet]
	TLV[TAG_1]	4-byte object identifier
	TLV[TAG_2]	2-byte key size in bits ( <a href="#">SE05x_RSABitLength_t</a> ) [Conditional: only when the object identifier is not in use yet]
	TLV[TAG_3]	P component [Conditional: only when the object identifier is in CRT mode and the key is generated externally and P1KeyPart is either P1_KEY_PAIR or P1_PRIVATE]
	TLV[TAG_4]	Q component [Conditional: only when the object identifier is in CRT mode and the key is generated externally and P1KeyPart is either P1_KEY_PAIR or P1_PRIVATE]
	TLV[TAG_5]	DP component [Conditional: only when the object identifier is in CRT mode and the key is generated externally and P1KeyPart is either P1_KEY_PAIR or P1_PRIVATE]
	TLV[TAG_6]	DQ component [Conditional: only when the object identifier is in CRT mode and the key is generated externally and P1KeyPart is either P1_KEY_PAIR or P1_PRIVATE]
	TLV[TAG_7]	INV_Q component [Conditional: only when the object identifier is in CRT mode and the key is generated externally and P1KeyPart is either P1_KEY_PAIR or P1_PRIVATE]
	TLV[TAG_8]	Public exponent
	TLV[TAG_9]	Private Key (non-CRT mode only)
	TLV[TAG_10]	Public Key (Modulus)
	TLV[TAG_11]	4-byte version [Optional]

**Return** The sm status.

#### Parameters

- [in] session\_ctx: The session context
- [in] policy: The policy
- [in] objectID: The object id
- [in] size: The size
- [in] p: The part p
- [in] pLen: The p length
- [in] q: The quarter
- [in] qLen: The quarter length

- [in] dp: The part dp
- [in] dpLen: The dp length
- [in] dq: The part dq
- [in] dqLen: The dq length
- [in] qInv: The quarter inv
- [in] qInvLen: The quarter inv length
- [in] pubExp: The pub exponent
- [in] pubExpLen: The pub exponent length
- [in] priv: The priv
- [in] privLen: The priv length
- [in] pubMod: The pub modifier
- [in] pubModLen: The pub modifier length
- [in] transient\_type: The transient type
- [in] key\_part: The key part
- [in] rsa\_format: The rsa format

## Function Se05x\_API\_WriteSymmKey

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h

## Function Documentation

smStatus\_t **Se05x\_API\_WriteSymmKey** (pSe05xSession\_t *session\_ctx*, pSe05xPolicy\_t *policy*, *SE05x\_MaxAttempts\_t* *maxAttempt*, uint32\_t *objectID*, *SE05x\_KeyID\_t* *kekID*, **const** uint8\_t *\*keyValue*, size\_t *keyValueLen*, **const** *SE05x\_INS\_t* *ins\_type*, **const** *SE05x\_SymmKeyType\_t* *type*)

Se05x\_API\_WriteSymmKey

Creates or writes an AES key, DES key or HMAC key, indicated by P1:

- P1\_AES
- P1\_DES
- P1\_HMAC

Users can pass RFC3394 wrapped keys by indicating the KEK in TLV[TAG\_2]. Note that RFC3394 required 8-byte aligned input, so this can only be used when the key has an 8-byte aligned length.

*Command to Applet*

Field	Value	Description
P1	See above	See <i>SE05x_P1_t</i>
P2	P2_DEFAULT	See <i>SE05x_P2_t</i>
Pay-load	TLV[TAG_POLICY]	Object array containing the object policy. [Optional: default policy applies] [Conditional: only when the object identifier is not in use yet]
	TLV[TAG_MAX_ATTEMPTS]	Number of attempts. If 0 is given, this means unlimited. [Optional: default unlimited] [Conditional: only when the object identifier is not in use yet and INS includes INS_AUTH_OBJECT; see AuthenticationObjectPolicies]
	TLV[TAG_1]	4-byte object identifier
	TLV[TAG_2]	4-byte KEK identifier [Conditional: only when the key value is RFC3394 wrapped]
	TLV[TAG_3]	Key value, either plain or RFC3394 wrapped.
	TLV[TAG_4]	Tag length for GCM/GMAC. Will only be used if the object is an AESKey. [Optional]
	TLV[TAG_11]	4-byte version [Optional]

**Return** The sm status.

#### Parameters

- [in] *session\_ctx*: The session context
- [in] *policy*: The policy
- [in] *maxAttempt*: The maximum attempt
- [in] *objectID*: The object id
- [in] *kekID*: The kek id
- [in] *keyValue*: The key value
- [in] *keyValueLen*: The key value length
- [in] *ins\_type*: The insert type
- [in] *type*: The type

#### Function Se05x\_API\_WriteUserID

- Defined in file *hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU\_apis.h*

#### Function Documentation

`smStatus_t Se05x_API_WriteUserID (pSe05xSession_t session_ctx, pSe05xPolicy_t policy, SE05x_MaxAttempts_t maxAttempt, uint32_t objectID, const uint8_t *userId, size_t userIdLen, const SE05x_AttestationType_t attestation_type)`

*Se05x\_API\_WriteUserID*

Creates a UserID object, setting the user identifier value. The policy defines the maximum number of attempts that can be performed as comparison.

*Command to Applet*

Field	Value	Description
P1	P1_USERID	See <a href="#">SE05x_P1_t</a>
P2	P2_DEFAULT	See <a href="#">SE05x_P2_t</a>
	TLV[TAG_POLICY]	Byte array containing the object policy. [Optional: default policy applies] [Conditional: only when the object identifier is not in use yet]
	TLV[TAG_MAXATTEMPTS]	Number of attempts. If 0 is given, this means unlimited. For pins, the maximum number of attempts must be smaller than 256. [Optional: default = 0] [Conditional: only when the object identifier is not in use yet and INS includes INS_AUTH_OBJECT; see -]
	TLV[TAG_1]	4-byte object identifier.
	TLV[TAG_2]	Byte array containing 4 to 16 bytes user identifier value.

**Return** The sm status.

#### Parameters

- [in] session\_ctx: The session context
- [in] policy: The policy
- [in] maxAttempt: The maximum attempt
- [in] objectID: The object id
- [in] userId: The user identifier
- [in] userIdLen: The user identifier length
- [in] attestation\_type: The attestation type

### Function se05x\_get\_sha\_algo

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

#### Function Documentation

*SE05x\_DigestMode\_t* **se05x\_get\_sha\_algo** (*sss\_algorithm\_t* algorithm)

### Function Se05x\_i2c\_master\_attst\_txn

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_types.h

#### Function Documentation

smStatus\_t **Se05x\_i2c\_master\_attst\_txn** (*sss\_session\_t* \*sess, *sss\_object\_t* \*keyObject, *SE05x\_I2CM\_cmd\_t* \*p, uint8\_t \*random\_attst, size\_t random\_attstLen, *SE05x\_AttestationAlgo\_t* attst\_algo, *SE05x\_TimeStamp\_t* \*ptimeStamp, size\_t \*timeStampLen, uint8\_t \*freshness, size\_t \*pfreshnessLen, uint8\_t \*chipId, size\_t \*pchipIdLen, uint8\_t \*signature, size\_t \*psignatureLen, uint8\_t noOfTags)

Se05x\_i2c\_master\_attst\_txn.

I2CM Read With Attestation

**Pre** p describes I2C master commands.

**Post** p contains execution state of I2C master commands, the I2C master commands can be overwritten to report on execution failure.

#### Parameters

- [in] sess: session identifier
- [in] keyObject: Keyobject which contains 4 byte attestaion KeyId
- [inout] p: Array of structure type capturing a sequence of i2c master cmd/rsp transactions.
- [in] random\_attst: 16-byte freshness random
- [in] random\_attstLen: length of freshness random
- [in] attst\_algo: 1 byte attestationAlgo
- [out] ptimeStamp: timestamp
- [out] timeStampLen: Length for timestamp
- [out] freshness: freshness (random)
- [out] pfreshnessLen: Length for freshness
- [out] chipId: unique chip Id
- [out] pchipIdLen: Length for chipId
- [out] signature: signature
- [out] psignatureLen: Length for signature
- [in] noOfTags: Amount of structures contained in p

#### Function Se05x\_i2c\_master\_txn

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_types.h

#### Function Documentation

smStatus\_t **Se05x\_i2c\_master\_txn** (*sss\_session\_t* \*sess, *SE05x\_I2CM\_cmd\_t* \*cmds, uint8\_t cmdLen)  
 Se05x\_i2c\_master\_txn.

I2CM Transaction

**Pre** p describes I2C master commands.

**Post** p contains execution state of I2C master commands, the I2C master commands can be overwritten to report on execution failure.

#### Parameters

- [in] sess: session identifier
- [inout] cmds: Array of structure type capturing a sequence of i2c master cmd/rsp transactions.
- [in] cmdLen: Amount of structures contained in cmds

## Function Se05x\_IsInvalidRangeOfUID

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU.h

## Function Documentation

bool **Se05x\_IsInvalidRangeOfUID** (uint32\_t uid)

## Function se05x\_sssKeyTypeLenToCurveId

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_types.h

## Function Documentation

uint32\_t **se05x\_sssKeyTypeLenToCurveId** (sss\_cipher\_type\_t keyType, size\_t keyBits)

## Function sss\_aead\_context\_free

- Defined in file\_sss\_inc\_fsl\_sss\_api.h

## Function Documentation

void **sss\_aead\_context\_free** (sss\_aead\_t \*context)  
AEAD context release. The function frees aead context.

### Parameters

- context: Pointer to aead context.

## Function sss\_aead\_context\_init

- Defined in file\_sss\_inc\_fsl\_sss\_api.h

## Function Documentation

sss\_status\_t **sss\_aead\_context\_init** (sss\_aead\_t \*context, sss\_session\_t \*session, sss\_object\_t \*key-Object, sss\_algorithm\_t algorithm, sss\_mode\_t mode)  
AEAD context init. The function initializes aead context with initial values.

**Return** Status of the operation

### Parameters

- context: Pointer to aead crypto context.
- session: Associate SSS session with aead context.
- keyObject: Associate SSS key object with aead context.
- algorithm: One of the aead algorithms defined by *sss\_algorithm\_t*.

- `mode`: One of the modes defined by `sss_mode_t`.

#### Return Value

- `kStatus_SSS_Success`: The operation has completed successfully.
- `kStatus_SSS_Fail`: The operation has failed.
- `kStatus_SSS_InvalidArgument`: One of the arguments is invalid for the function to execute.

### Function `sss_aead_finish`

- Defined in file `_sss_inc_fsl_sss_api.h`

#### Function Documentation

`sss_status_t sss_aead_finish(sss_aead_t *context, const uint8_t *srcData, size_t srcLen, uint8_t *destData, size_t *destLen, uint8_t *tag, size_t *tagLen)`

Finalize AEAD. The function processes data that has not been processed by previous calls to `sss_aead_update()` as well as `srcData`. It finalizes the AEAD operations and computes the tag (encryption) or compares the computed tag with the tag supplied in the parameter (decryption).

**Return** Status of the operation

#### Parameters

- `context`: Pointer to aead crypto context.
- `srcData`: Buffer containing final chunk of input data.
- `srcLen`: Length of final chunk of input data in bytes.
- `destData`: Buffer containing output data.
- `[inout] destLen`: Length of output data in bytes. Buffer length on entry, reflects actual output size on return.
- `tag`: Encryption: Output buffer filled with computed tag. Decryption: Input buffer filled with received tag.
- `tagLen`: Length of the computed or received tag in bytes. For AES-GCM it must be 4,8,12,13,14,15 or 16. For AES-CCM it must be 4,6,8,10,12,14 or 16.

#### Return Value

- `kStatus_SSS_Success`: The operation has completed successfully.
- `kStatus_SSS_Fail`: The operation has failed.
- `kStatus_SSS_InvalidArgument`: One of the arguments is invalid for the function to execute.

## Function `sss_aead_init`

- Defined in file `_sss_inc_fsl_sss_api.h`

## Function Documentation

*sss\_status\_t* **sss\_aead\_init** (*sss\_aead\_t* \*context, uint8\_t \*nonce, size\_t nonceLen, size\_t tagLen, size\_t aadLen, size\_t payloadLen)

AEAD init. The function starts the aead operation.

**Return** Status of the operation

### Parameters

- context: Pointer to aead crypto context.
- nonce: The operation nonce or IV.
- nonceLen: The length of nonce in bytes. For AES-GCM it must be  $\geq 1$ . For AES-CCM it must be 7, 8, 9, 10, 11, 12, or 13.
- tagLen: Length of the computed or received tag in bytes. For AES-GCM it must be 4,8,12,13,14,15 or 16. For AES-CCM it must be 4,6,8,10,12,14 or 16.
- aadLen: Input size in bytes of AAD. Used only for AES-CCM. Ignored for AES-GCM.
- payloadLen: Length in bytes of the payload. Used only for AES-CCM. Ignored for AES-GCM.

### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.

## Function `sss_aead_one_go`

- Defined in file `_sss_inc_fsl_sss_api.h`

## Function Documentation

*sss\_status\_t* **sss\_aead\_one\_go** (*sss\_aead\_t* \*context, **const** uint8\_t \*srcData, uint8\_t \*destData, size\_t size, uint8\_t \*nonce, size\_t nonceLen, **const** uint8\_t \*aad, size\_t aadLen, uint8\_t \*tag, size\_t \*tagLen)

AEAD in one blocking function call. The function blocks current thread until the operation completes or an error occurs.

**Return** Status of the operation

### Parameters

- context: Pointer to aead crypto context.
- srcData: Buffer containing the input data.
- destData: Buffer containing the output data.
- size: Size of input and output data buffer in bytes.
- nonce: The operation nonce or IV.



- `nonceLen`: The length of nonce in bytes. For AES-GCM it must be  $\geq 1$ . For AES-CCM it must be 7, 8, 9, 10, 11, 12, or 13.
- `aad`: Input additional authentication data AAD
- `aadLen`: Input size in bytes of AAD
- `tag`: Encryption: Output buffer filled with computed tag Decryption: Input buffer filled with received tag
- `tagLen`: Length of the tag in bytes. For AES-GCM it must be 4,8,12,13,14,15 or 16. For AES-CCM it must be 4,6,8,10,12,14 or 16.

#### Return Value

- `kStatus_SSS_Success`: The operation has completed successfully.
- `kStatus_SSS_Fail`: The operation has failed.

#### Function `sss_aead_update`

- Defined in file `_sss_inc_fsl_sss_api.h`

#### Function Documentation

`sss_status_t sss_aead_update(sss_aead_t *context, const uint8_t *srcData, size_t srcLen, uint8_t *destData, size_t *destLen)`

AEAD data update. Feeds a new chunk of the data payload. Input data does not have to be a multiple of block size. Subsequent calls to this function are possible. Unless one or more calls of this function have supplied sufficient input data, no output is generated. The integration check is done by `sss_aead_finish()`. Until then it is not sure if the decrypt data is authentic.

**Return** Status of the operation

#### Parameters

- `context`: Pointer to aead crypto context.
- `srcData`: Buffer containing the input data.
- `srcLen`: Length of the input data in bytes.
- `destData`: Buffer containing the output data.
- `[inout] destLen`: Length of the output data in bytes. Buffer length on entry, reflects actual output size on return.

#### Return Value

- `kStatus_SSS_Success`: The operation has completed successfully.
- `kStatus_SSS_Fail`: The operation has failed.
- `kStatus_SSS_InvalidArgument`: One of the arguments is invalid for the function to execute.

## Function `sss_aead_update_aad`

- Defined in file\_sss\_inc\_fsl\_sss\_api.h

### Function Documentation

*sss\_status\_t* **sss\_aead\_update\_aad**(*sss\_aead\_t* \*context, **const** uint8\_t \*aadData, size\_t aadDataLen)  
Feeds a new chunk of the AAD. Subsequent calls of this function are possible.

**Return** Status of the operation

#### Parameters

- context: Pointer to aead crypto context
- aadData: Input buffer containing the chunk of AAD
- aadDataLen: Length of the AAD data in bytes.

#### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.
- *kStatus\_SSS\_InvalidArgument*: One of the arguments is invalid for the function to execute.

## Function `sss_asymmetric_context_free`

- Defined in file\_sss\_inc\_fsl\_sss\_api.h

### Function Documentation

void **sss\_asymmetric\_context\_free**(*sss\_asymmetric\_t* \*context)  
Asymmetric context release. The function frees asymmetric context.

#### Parameters

- context: Pointer to asymmetric context.

## Function `sss_asymmetric_context_init`

- Defined in file\_sss\_inc\_fsl\_sss\_api.h

## Function Documentation

*sss\_status\_t* **sss\_asymmetric\_context\_init** (*sss\_asymmetric\_t* \*context, *sss\_session\_t* \*session, *sss\_object\_t* \*keyObject, *sss\_algorithm\_t* algorithm, *sss\_mode\_t* mode)

Asymmetric context init. The function initializes asymmetric context with initial values.

**Return** Status of the operation

### Parameters

- context: Pointer to asymmetric crypto context.
- session: Associate SSS session with asymmetric context.
- keyObject: Associate SSS key object with asymmetric context.
- algorithm: One of the asymmetric algorithms defined by *sss\_algorithm\_t*.
- mode: One of the modes defined by *sss\_mode\_t*.

### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.
- *kStatus\_SSS\_InvalidArgument*: One of the arguments is invalid for the function to execute.

## Function *sss\_asymmetric\_decrypt*

- Defined in file\_sss\_inc\_fsl\_sss\_api.h

## Function Documentation

*sss\_status\_t* **sss\_asymmetric\_decrypt** (*sss\_asymmetric\_t* \*context, **const** uint8\_t \*srcData, size\_t srcLen, uint8\_t \*destData, size\_t \*destLen)

Asymmetric decryption The function uses asymmetric algorithm to decrypt data. Private key portion of a key pair is used for decryption.

**Return** Status of the operation

### Parameters

- context: Pointer to asymmetric context.
- srcData: Input buffer
- srcLen: Length of the input in bytes
- destData: Output buffer
- destLen: Length of the output in bytes

### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.
- *kStatus\_SSS\_InvalidArgument*: One of the arguments is invalid for the function to execute.

## Function `sss_asymmetric_encrypt`

- Defined in file `sss_inc_fsl_sss_api.h`

### Function Documentation

*sss\_status\_t* **sss\_asymmetric\_encrypt** (*sss\_asymmetric\_t* \*context, **const** uint8\_t \*srcData, size\_t srcLen, uint8\_t \*destData, size\_t destLen)

Asymmetric encryption The function uses asymmetric algorithm to encrypt data. Public key portion of a key pair is used for encryption.

**Return** Status of the operation

#### Parameters

- context: Pointer to asymmetric context.
- srcData: Input buffer
- srcLen: Length of the input in bytes
- destData: Output buffer
- destLen: Length of the output in bytes

#### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.
- *kStatus\_SSS\_InvalidArgument*: One of the arguments is invalid for the function to execute.

## Function `sss_asymmetric_sign_digest`

- Defined in file `sss_inc_fsl_sss_api.h`

### Function Documentation

*sss\_status\_t* **sss\_asymmetric\_sign\_digest** (*sss\_asymmetric\_t* \*context, uint8\_t \*digest, size\_t digestLen, uint8\_t \*signature, size\_t signatureLen)

Asymmetric signature of a message digest The function signs a message digest.

**Return** Status of the operation

#### Parameters

- context: Pointer to asymmetric context.
- digest: Input buffer containing the input message digest
- digestLen: Length of the digest in bytes
- signature: Output buffer written with the signature of the digest
- signatureLen: Length of the signature in bytes

#### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.

- *kStatus\_SSS\_Fail*: The operation has failed.
- *kStatus\_SSS\_InvalidArgument*: One of the arguments is invalid for the function to execute.

## Function `sss_asymmetric_verify_digest`

- Defined in `file_sss_inc_fsl_sss_api.h`

### Function Documentation

*sss\_status\_t* **sss\_asymmetric\_verify\_digest** (*sss\_asymmetric\_t* \*context, uint8\_t \*digest, size\_t digestLen, uint8\_t \*signature, size\_t signatureLen)

Asymmetric verify of a message digest The function verifies a message digest.

**Return** Status of the operation

#### Parameters

- context: Pointer to asymmetric context.
- digest: Input buffer containing the input message digest
- digestLen: Length of the digest in bytes
- signature: Input buffer containing the signature to verify
- signatureLen: Length of the signature in bytes

#### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.
- *kStatus\_SSS\_InvalidArgument*: One of the arguments is invalid for the function to execute.

## Function `sss_cipher_crypt_ctr`

- Defined in `file_sss_inc_fsl_sss_api.h`

### Function Documentation

*sss\_status\_t* **sss\_cipher\_crypt\_ctr** (*sss\_symmetric\_t* \*context, const uint8\_t \*srcData, uint8\_t \*destData, size\_t size, uint8\_t \*initialCounter, uint8\_t \*lastEncryptedCounter, size\_t \*szLeft)

Symmetric AES in Counter mode in one blocking function call. The function blocks current thread until the operation completes or an error occurs.

**Return** Status of the operation

#### Parameters

- context: Pointer to symmetric crypto context.
- srcData: Buffer containing the input data.
- destData: Buffer containing the output data.

- `size`: Size of source and destination data buffers in bytes.
- `[inout] initialCounter`: Input counter (updates on return)
- `[out] lastEncryptedCounter`: Output cipher of last counter, for chained CTR calls. NULL can be passed if chained calls are not used.
- `[out] szLeft`: Output number of bytes in left unused in `lastEncryptedCounter` block. NULL can be passed if chained calls are not used.

#### Return Value

- *`kStatus_SSS_Success`*: The operation has completed successfully.
- *`kStatus_SSS_Fail`*: The operation has failed.

### Function `sss_cipher_finish`

- Defined in `file_sss_inc_fsl_sss_api.h`

#### Function Documentation

*`sss_status_t`* **`sss_cipher_finish`** (*`sss_symmetric_t`* \**context*, **const** *uint8\_t* \**srcData*, *size\_t* *srcLen*, *uint8\_t* \**destData*, *size\_t* \**destLen*)

Symmetric cipher finalize.

**Return** Status of the operation

#### Parameters

- *context*: Pointer to symmetric crypto context.
- *srcData*: Buffer containing final chunk of input data.
- *srcLen*: Length of final chunk of input data in bytes.
- *destData*: Buffer containing output data.
- `[inout] destLen`: Length of output data in bytes. Buffer length on entry, reflects actual output size on return.

#### Return Value

- *`kStatus_SSS_Success`*: The operation has completed successfully.
- *`kStatus_SSS_Fail`*: The operation has failed.
- *`kStatus_SSS_InvalidArgument`*: One of the arguments is invalid for the function to execute.

### Function `sss_cipher_init`

- Defined in `file_sss_inc_fsl_sss_api.h`

## Function Documentation

*sss\_status\_t* **sss\_cipher\_init** (*sss\_symmetric\_t* \*context, uint8\_t \*iv, size\_t ivLen)

Symmetric cipher init. The function starts the symmetric cipher operation.

**Return** Status of the operation

### Parameters

- context: Pointer to symmetric crypto context.
- iv: Buffer containing the symmetric operation Initialization Vector.
- ivLen: Length of the Initialization Vector in bytes.

### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.

## Function sss\_cipher\_one\_go

- Defined in file\_sss\_inc\_fsl\_sss\_api.h

## Function Documentation

*sss\_status\_t* **sss\_cipher\_one\_go** (*sss\_symmetric\_t* \*context, uint8\_t \*iv, size\_t ivLen, **const** uint8\_t \*srcData, uint8\_t \*destData, size\_t dataLen)

Symmetric cipher in one blocking function call. The function blocks current thread until the operation completes or an error occurs.

**Return** Status of the operation

### Parameters

- context: Pointer to symmetric crypto context.
- iv: Buffer containing the symmetric operation Initialization Vector.
- ivLen: Length of the Initialization Vector in bytes.
- srcData: Buffer containing the input data.
- destData: Buffer containing the output data.
- dataLen: Size of input and output data buffer in bytes.

### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.

## Function `sss_cipher_update`

- Defined in file `sss_inc_fsl_sss_api.h`

### Function Documentation

*sss\_status\_t* **sss\_cipher\_update** (*sss\_symmetric\_t* \*context, **const** uint8\_t \*srcData, size\_t srcLen, uint8\_t \*destData, size\_t \*destLen)

Symmetric cipher update. Input data does not have to be a multiple of block size. Subsequent calls to this function are possible. Unless one or more calls of this function have supplied sufficient input data, no output is generated. The cipher operation is finalized with a call to *sss\_cipher\_finish()*.

**Return** Status of the operation

#### Parameters

- context: Pointer to symmetric crypto context.
- srcData: Buffer containing the input data.
- srcLen: Length of the input data in bytes.
- destData: Buffer containing the output data.
- [inout] destLen: Length of the output data in bytes. Buffer length on entry, reflects actual output size on return.

#### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.
- *kStatus\_SSS\_InvalidArgument*: One of the arguments is invalid for the function to execute.

## Function `sss_derive_key_context_free`

- Defined in file `sss_inc_fsl_sss_api.h`

### Function Documentation

void **sss\_derive\_key\_context\_free** (*sss\_derive\_key\_t* \*context)

Derive key context release. The function frees derive key context.

#### Parameters

- context: Pointer to derive key context.



## Function `sss_derive_key_context_init`

- Defined in file `_sss_inc_fsl_sss_api.h`

### Function Documentation

`sss_status_t sss_derive_key_context_init (sss_derive_key_t *context, sss_session_t *session, sss_object_t *keyObject, sss_algorithm_t algorithm, sss_mode_t mode)`

Derive key context init. The function initializes derive key context with initial values.

**Return** Status of the operation

#### Parameters

- `context`: Pointer to derive key context.
- `session`: Associate SSS session with the derive key context.
- `keyObject`: Associate SSS key object with the derive key context.
- `algorithm`: One of the derive key algorithms defined by `sss_algorithm_t`.
- `mode`: One of the modes defined by `sss_mode_t`.

#### Return Value

- `kStatus_SSS_Success`: The operation has completed successfully.
- `kStatus_SSS_Fail`: The operation has failed.
- `kStatus_SSS_InvalidArgument`: One of the arguments is invalid for the function to execute.

## Function `sss_derive_key_dh`

- Defined in file `_sss_inc_fsl_sss_api.h`

### Function Documentation

`sss_status_t sss_derive_key_dh (sss_derive_key_t *context, sss_object_t *otherPartyKeyObject, sss_object_t *derivedKeyObject)`

Asymmetric key derivation Diffie-Hellmann The function cryptographically derives a key from another key. For example Diffie-Hellmann.

**Return** Status of the operation

#### Parameters

- `context`: Pointer to derive key context.
- `otherPartyKeyObject`: Public key of the other party in the Diffie-Hellmann algorithm
- `[inout] derivedKeyObject`: Reference to a derived key

#### Return Value

- `kStatus_SSS_Success`: The operation has completed successfully.
- `kStatus_SSS_Fail`: The operation has failed.

- *kStatus\_SSS\_InvalidArgument*: One of the arguments is invalid for the function to execute.

## Function `sss_derive_key_go`

- Defined in `file_sss_inc_fsl_sss_api.h`

## Function Documentation

```
sss_status_t sss_derive_key_go(sss_derive_key_t *context, const uint8_t *saltData, size_t saltLen,
 const uint8_t *info, size_t infoLen, sss_object_t *derivedKeyObject,
 uint16_t deriveDataLen, uint8_t *hkdfOutput, size_t *hkdfOutputLen)
```

Symmetric key derivation The function cryptographically derives a key from another key. For example MIFARE key derivation, PRF, HKDF-Extract.

**Return** Status of the operation

### Parameters

- context: Pointer to derive key context.
- saltData: Input data buffer, typically with some random data.
- saltLen: Length of saltData buffer in bytes.
- info: Input data buffer, typically with some fixed info.
- infoLen: Length of info buffer in bytes.
- [inout] derivedKeyObject: Reference to a derived key
- deriveDataLen: **TODO** Document this
- hkdfOutput: **TODO** Document this
- hkdfOutputLen: **TODO** Document this

### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.
- *kStatus\_SSS\_InvalidArgument*: One of the arguments is invalid for the function to execute.

## Function `sss_derive_key_one_go`

- Defined in `file_sss_inc_fsl_sss_api.h`

## Function Documentation

*sss\_status\_t* **sss\_derive\_key\_one\_go** (*sss\_derive\_key\_t* \*context, **const** uint8\_t \*saltData, size\_t saltLen, **const** uint8\_t \*info, size\_t infoLen, *sss\_object\_t* \*derivedKeyObject, uint16\_t deriveDataLen)

Symmetric key derivation (replaces the deprecated function *sss\_derive\_key\_go*) The function cryptographically derives a key from another key. For example MIFARE key derivation, PRF, HKDF-Extract-Expand, HKDF-Expand. Refer to *sss\_derive\_key\_sobj\_one\_go* in case the Salt is available as a key object.

**Return** Status of the operation

### Parameters

- context: Pointer to derive key context.
- saltData: Input data buffer, typically with some random data.
- saltLen: Length of saltData buffer in bytes.
- info: Input data buffer, typically with some fixed info.
- infoLen: Length of info buffer in bytes.
- [inout] derivedKeyObject: Reference to a derived key
- [in] deriveDataLen: Expected length of derived key.

### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.
- *kStatus\_SSS\_InvalidArgument*: One of the arguments is invalid for the function to execute.

## Function *sss\_derive\_key\_sobj\_one\_go*

- Defined in file\_sss\_inc\_fsl\_sss\_api.h

## Function Documentation

*sss\_status\_t* **sss\_derive\_key\_sobj\_one\_go** (*sss\_derive\_key\_t* \*context, *sss\_object\_t* \*saltKeyObject, **const** uint8\_t \*info, size\_t infoLen, *sss\_object\_t* \*derivedKeyObject, uint16\_t deriveDataLen)

Symmetric key derivation (salt in key object) Refer to *sss\_derive\_key\_one\_go* in case the salt is not available as a key object.

**Return** Status of the operation

### Parameters

- context: Pointer to derive key context
- saltKeyObject: Reference to salt. The salt key object must reside in the same keystore as the derive key context.
- [in] info: Input data buffer, typically with some fixed info.
- [in] infoLen: Length of info buffer in bytes.
- derivedKeyObject: Reference to a derived key

- [in] `deriveDataLen`: The derive data length

#### Return Value

- *`kStatus_SSS_Success`*: The operation has completed successfully.
- *`kStatus_SSS_Fail`*: The operation has failed.
- *`kStatus_SSS_InvalidArgument`*: One of the arguments is invalid for the function to execute.

### Function `sss_digest_context_free`

- Defined in `file_sss_inc_fsl_sss_api.h`

#### Function Documentation

void **`sss_digest_context_free`** (*`sss_digest_t`* \*context)  
Digest context release. The function frees digest context.

#### Parameters

- `context`: Pointer to digest context.

### Function `sss_digest_context_init`

- Defined in `file_sss_inc_fsl_sss_api.h`

#### Function Documentation

*`sss_status_t`* **`sss_digest_context_init`** (*`sss_digest_t`* \*context, *`sss_session_t`* \*session, *`sss_algorithm_t`* algorithm, *`sss_mode_t`* mode)  
Digest context init. The function initializes digest context with initial values.

**Return** Status of the operation

#### Parameters

- `context`: Pointer to digest context.
- `session`: Associate SSS session with digest context.
- `algorithm`: One of the digest algorithms defined by *`sss_algorithm_t`*.
- `mode`: One of the modes defined by *`sss_mode_t`*.

#### Return Value

- *`kStatus_SSS_Success`*: The operation has completed successfully.
- *`kStatus_SSS_Fail`*: The operation has failed.
- *`kStatus_SSS_InvalidArgument`*: One of the arguments is invalid for the function to execute.

## Function `sss_digest_finish`

- Defined in `file_sss_inc_fsl_sss_api.h`

### Function Documentation

*sss\_status\_t* **sss\_digest\_finish** (*sss\_digest\_t* \*context, uint8\_t \*digest, size\_t \*digestLen)

Finish digest for a message. The function blocks current thread until the operation completes or an error occurs.

**Return** Status of the operation

#### Parameters

- context: Pointer to digest context.
- digest: Output message digest
- digestLen: Message digest byte length

#### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.

## Function `sss_digest_init`

- Defined in `file_sss_inc_fsl_sss_api.h`

### Function Documentation

*sss\_status\_t* **sss\_digest\_init** (*sss\_digest\_t* \*context)

Init digest for a message. The function blocks current thread until the operation completes or an error occurs.

**Return** Status of the operation

#### Parameters

- context: Pointer to digest context.

#### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.

## Function `sss_digest_one_go`

- Defined in `file_sss_inc_fsl_sss_api.h`

## Function Documentation

*sss\_status\_t* **sss\_digest\_one\_go** (*sss\_digest\_t* \*context, **const** uint8\_t \*message, size\_t messageLen, uint8\_t \*digest, size\_t \*digestLen)

Message digest in one blocking function call. The function blocks current thread until the operation completes or an error occurs.

**Return** Status of the operation

### Parameters

- context: Pointer to digest context.
- message: Input message
- messageLen: Length of the input message in bytes
- digest: Output message digest
- digestLen: Message digest byte length

### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.

## Function **sss\_digest\_update**

- Defined in file\_sss\_inc\_fsl\_sss\_api.h

## Function Documentation

*sss\_status\_t* **sss\_digest\_update** (*sss\_digest\_t* \*context, **const** uint8\_t \*message, size\_t messageLen)

Update digest for a message.

The function blocks current thread until the operation completes or an error occurs.

**Return** Status of the operation

### Parameters

- context: Pointer to digest context.
- message: Buffer with a message chunk.
- messageLen: Length of the input buffer in bytes.

### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.

## Function `sss_key_object_allocate_handle`

- Defined in file `sss_inc_fsl_sss_api.h`

### Function Documentation

`sss_status_t sss_key_object_allocate_handle` (`sss_object_t` \*keyObject, `uint32_t` keyId, `sss_key_part_t` keyPart, `sss_cipher_type_t` cipherType, `size_t` keyByteLenMax, `uint32_t` options)

Allocate / pre-provision memory for new key.

This API allows underlying cryptographic subsystems to perform preconditions of before creating any cryptographic key object.

**Return** Status of object allocation.

#### Parameters

- [inout] keyObject: The object If required, update implementation defined values inside the keyObject
- keyId: External Key ID. Later on this may be used by `sss_key_object_get_handle`
- keyPart: See `sss_key_part_t`
- cipherType: See `sss_cipher_type_t`
- keyByteLenMax: Maximum storage this type of key may need. For systems that have their own internal allocation table this would help
- options: 0 = Persistent Key (Default) or Transient Key. See `sss_key_object_mode_t`

## Function `sss_key_object_free`

- Defined in file `sss_inc_fsl_sss_api.h`

### Function Documentation

void `sss_key_object_free` (`sss_object_t` \*keyObject)

Destructor for the key object. The function frees key object context.

#### Parameters

- keyObject: Pointer to key object context.

## Function `sss_key_object_get_access`

- Defined in `file_sss_inc_fsl_sss_api.h`

### Function Documentation

*sss\_status\_t* **sss\_key\_object\_get\_access** (*sss\_object\_t* \*keyObject, uint32\_t \*access)

Check what are access restrictions on an object

#### Return

#### Parameters

- keyObject: Object
- access: What is permitted

## Function `sss_key_object_get_handle`

- Defined in `file_sss_inc_fsl_sss_api.h`

### Function Documentation

*sss\_status\_t* **sss\_key\_object\_get\_handle** (*sss\_object\_t* \*keyObject, uint32\_t keyId)

Get handle to an existing allocated/provisioned/created Object.

See *sss\_key\_object\_allocate\_handle*.

After calling this API, Ideally keyObject should become equivalent to as set after the calling of *sss\_key\_object\_allocate\_handle* api.

**Return** The sss status.

#### Parameters

- keyObject: The key object
- [in] keyId: The key identifier

## Function `sss_key_object_get_purpose`

- Defined in `file_sss_inc_fsl_sss_api.h`

### Function Documentation

*sss\_status\_t* **sss\_key\_object\_get\_purpose** (*sss\_object\_t* \*keyObject, *sss\_mode\_t* \*purpose)

Check what is purpose restrictions on an object

#### Return

#### Parameters

- keyObject: Object to be checked



- purpose: Know what is permitted.

### Function `sss_key_object_get_user`

- Defined in file `sss_inc_fsl_sss_api.h`

#### Function Documentation

*sss\_status\_t* **sss\_key\_object\_get\_user** (*sss\_object\_t* \*keyObject, uint32\_t \*user)  
get attributes

### Function `sss_key_object_init`

- Defined in file `sss_inc_fsl_sss_api.h`

#### Function Documentation

*sss\_status\_t* **sss\_key\_object\_init** (*sss\_object\_t* \*keyObject, *sss\_key\_store\_t* \*keyStore)

Constructor for a key object data structure The function initializes keyObject data structure and associates it with a key store in which the plain key and other attributes are stored.

**Return** Status of the operation

#### Parameters

- keyObject:
- keyStore:

#### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.
- *kStatus\_SSS\_InvalidArgument*: One of the arguments is invalid for the function to execute.

### Function `sss_key_object_set_access`

- Defined in file `sss_inc_fsl_sss_api.h`

#### Function Documentation

*sss\_status\_t* **sss\_key\_object\_set\_access** (*sss\_object\_t* \*keyObject, uint32\_t access, uint32\_t options)  
Assign access permissions to a key object.

#### Parameters

- keyObject: the object where permission restrictions are applied
- access: Logical OR of read, write, delete, use, change attributes defined by enum `_sss_access_permission`.

- `options`: Transient or persistent update. Allows for transient update of persistent attributes.

### Function `sss_key_object_set_eccgfp_group`

- Defined in `file_sss_inc_fsl_sss_api.h`

#### Function Documentation

*sss\_status\_t* **sss\_key\_object\_set\_eccgfp\_group** (*sss\_object\_t* \*keyObject, *sss\_eccgfp\_group\_t* \*group)

Set elliptic curve domain parameters over Fp for a key object.

When the key object is a reference to one of ECC Private, ECC Public or ECC Pair key types, this function shall be used to specify the exact domain parameters prior to using the key object for ECDSA or ECDH algorithms.

#### Parameters

- `keyObject`: The destination key object
- `group`: Pointer to elliptic curve domain parameters over Fp (sextuple p,a,b,G,n,h)

### Function `sss_key_object_set_purpose`

- Defined in `file_sss_inc_fsl_sss_api.h`

#### Function Documentation

*sss\_status\_t* **sss\_key\_object\_set\_purpose** (*sss\_object\_t* \*keyObject, *sss\_mode\_t* purpose, *uint32\_t* options)

Assign purpose to a key object.

#### Parameters

- `keyObject`: the object where permission restrictions are applied
- `purpose`: Usage of the key.
- `options`: Transient or persistent update. Allows for transient update of persistent attributes.

### Function `sss_key_object_set_user`

- Defined in `file_sss_inc_fsl_sss_api.h`

## Function Documentation

*sss\_status\_t* **sss\_key\_object\_set\_user** (*sss\_object\_t* \*keyObject, uint32\_t user, uint32\_t options)

Assign user to a key object.

### Parameters

- keyObject: the object where permission restrictions are applied
- user: Assign User id for a key object. The user is kept in the key store along with the key data and other properties.
- options: Transient or persistent update. Allows for transient update of persistent attributes.

## Function sss\_key\_store\_allocate

- Defined in file\_sss\_inc\_fsl\_sss\_api.h

## Function Documentation

*sss\_status\_t* **sss\_key\_store\_allocate** (*sss\_key\_store\_t* \*keyStore, uint32\_t keyStoreId)

Get handle to key store. If the key store already exists, nothing is allocated. If the key store does not exist, new empty key store is created and initialized. Key store context structure is updated with actual information.

### Parameters

- [out] keyStore: Pointer to key store context. Key store context is updated on function return.
- keyStoreId: Implementation specific ID, can be used in case security subsystem manages multiple different key stores.

## Function sss\_key\_store\_context\_free

- Defined in file\_sss\_inc\_fsl\_sss\_api.h

## Function Documentation

void **sss\_key\_store\_context\_free** (*sss\_key\_store\_t* \*keyStore)

Destructor for the key store context.

## Function sss\_key\_store\_context\_init

- Defined in file\_sss\_inc\_fsl\_sss\_api.h

## Function Documentation

*sss\_status\_t* **sss\_key\_store\_context\_init** (*sss\_key\_store\_t* \*keyStore, *sss\_session\_t* \*session)

Constructor for the key store context data structure.

### Parameters

- [out] keyStore: Pointer to key store context. Key store context is updated on function return.
- session: Session context.

## Function **sss\_key\_store\_erase\_key**

- Defined in file\_sss\_inc\_fsl\_sss\_api.h

## Function Documentation

*sss\_status\_t* **sss\_key\_store\_erase\_key** (*sss\_key\_store\_t* \*keyStore, *sss\_object\_t* \*keyObject)

Delete / destroy allocated keyObject .

**Return** The sss status.

### Parameters

- keyStore: The key store
- keyObject: The key object to be deleted

## Function **sss\_key\_store\_freeze\_key**

- Defined in file\_sss\_inc\_fsl\_sss\_api.h

## Function Documentation

*sss\_status\_t* **sss\_key\_store\_freeze\_key** (*sss\_key\_store\_t* \*keyStore, *sss\_object\_t* \*keyObject)

The referenced key cannot be updated any more.

**Return** The sss status.

### Parameters

- keyStore: The key store
- keyObject: The key object to be locked / frozen.

### Function `sss_key_store_generate_key`

- Defined in file `sss_inc_fsl_sss_api.h`

#### Function Documentation

*sss\_status\_t* **sss\_key\_store\_generate\_key** (*sss\_key\_store\_t* \*keyStore, *sss\_object\_t* \*keyObject, size\_t keyBitLen, void \*options)

This function generates key[] in the destination key store.

### Function `sss_key_store_get_key`

- Defined in file `sss_inc_fsl_sss_api.h`

#### Function Documentation

*sss\_status\_t* **sss\_key\_store\_get\_key** (*sss\_key\_store\_t* \*keyStore, *sss\_object\_t* \*keyObject, uint8\_t \*data, size\_t \*dataLen, size\_t \*pKeyBitLen)

This function exports plain key[] from key store (if constraints and user id allows reading)

### Function `sss_key_store_load`

- Defined in file `sss_inc_fsl_sss_api.h`

#### Function Documentation

*sss\_status\_t* **sss\_key\_store\_load** (*sss\_key\_store\_t* \*keyStore)

Load from persistent memory to cached objects.

### Function `sss_key_store_open_key`

- Defined in file `sss_inc_fsl_sss_api.h`

#### Function Documentation

*sss\_status\_t* **sss\_key\_store\_open\_key** (*sss\_key\_store\_t* \*keyStore, *sss\_object\_t* \*keyObject)

Access key store using one more level of encryption.

e.g. Access keys / encryption key during storage

**Return** The sss status.

#### Parameters

- keyStore: The key store
- keyObject: The key object that is to be used as a KEK (Key Encryption Key)

## Function `sss_key_store_save`

- Defined in file\_sss\_inc\_fsl\_sss\_api.h

### Function Documentation

*sss\_status\_t* **sss\_key\_store\_save** (*sss\_key\_store\_t* \*keyStore)

Save all cached persistent objects to persistent memory.

## Function `sss_key_store_set_key`

- Defined in file\_sss\_inc\_fsl\_sss\_api.h

### Function Documentation

*sss\_status\_t* **sss\_key\_store\_set\_key** (*sss\_key\_store\_t* \*keyStore, *sss\_object\_t* \*keyObject, **const** *uint8\_t* \*data, *size\_t* dataLen, *size\_t* keyBitLen, *void* \*options, *size\_t* optionsLen)

This function moves data[] from memory to the destination key store.

#### Return

#### Parameters

- keyStore: Key store context
- keyObject: Reference to a key and it's properties
- data: Data to be stored in Key. When setting ecc private key only, do not include key header.
- dataLen: Length of the data
- keyBitLen: Crypto algorithm key bit length
- options: Pointer to implementation specific options
- optionsLen: Length of the options in bytes

## Function `sss_mac_context_free`

- Defined in file\_sss\_inc\_fsl\_sss\_api.h

### Function Documentation

*void* **sss\_mac\_context\_free** (*sss\_mac\_t* \*context)

MAC context release. The function frees mac context.

#### Parameters

- context: Pointer to mac context.

## Function `sss_mac_context_init`

- Defined in `file_sss_inc_fsl_sss_api.h`

### Function Documentation

*sss\_status\_t* **sss\_mac\_context\_init** (*sss\_mac\_t* \*context, *sss\_session\_t* \*session, *sss\_object\_t* \*keyObject, *sss\_algorithm\_t* algorithm, *sss\_mode\_t* mode)

MAC context init. The function initializes mac context with initial values.

**Return** Status of the operation

#### Parameters

- context: Pointer to mac context.
- session: Associate SSS session with mac context.
- keyObject: Associate SSS key object with mac context.
- algorithm: One of the mac algorithms defined by *sss\_algorithm\_t*.
- mode: One of the modes defined by *sss\_mode\_t*.

#### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.
- *kStatus\_SSS\_InvalidArgument*: One of the arguments is invalid for the function to execute.

## Function `sss_mac_finish`

- Defined in `file_sss_inc_fsl_sss_api.h`

### Function Documentation

*sss\_status\_t* **sss\_mac\_finish** (*sss\_mac\_t* \*context, *uint8\_t* \*mac, *size\_t* \*macLen)

Finish mac for a message. The function blocks current thread until the operation completes or an error occurs.

**Return** Status of the operation

#### Parameters

- context: Pointer to mac context.
- mac: Output message MAC
- macLen: Computed MAC byte length

#### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.

## Function `sss_mac_init`

- Defined in `file_sss_inc_fsl_sss_api.h`

### Function Documentation

*sss\_status\_t* **sss\_mac\_init** (*sss\_mac\_t* \*context)

Init mac for a message. The function blocks current thread until the operation completes or an error occurs.

**Return** Status of the operation

#### Parameters

- context: Pointer to mac context.

#### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.

## Function `sss_mac_one_go`

- Defined in `file_sss_inc_fsl_sss_api.h`

### Function Documentation

*sss\_status\_t* **sss\_mac\_one\_go** (*sss\_mac\_t* \*context, **const** *uint8\_t* \*message, *size\_t* messageLen, *uint8\_t* \*mac, *size\_t* \*macLen)

Message MAC in one blocking function call. The function blocks current thread until the operation completes or an error occurs.

**Return** Status of the operation

#### Parameters

- context: Pointer to mac context.
- message: Input message
- messageLen: Length of the input message in bytes
- mac: Output message MAC
- macLen: Computed MAC byte length

#### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.



## Function `sss_mac_update`

- Defined in file `_sss_inc_fsl_sss_api.h`

### Function Documentation

`sss_status_t sss_mac_update (sss_mac_t *context, const uint8_t *message, size_t messageLen)`  
Update mac for a message.

The function blocks current thread until the operation completes or an error occurs.

**Return** Status of the operation

#### Parameters

- `context`: Pointer to mac context.
- `message`: Buffer with a message chunk.
- `messageLen`: Length of the input buffer in bytes.

#### Return Value

- `kStatus_SSS_Success`: The operation has completed successfully.
- `kStatus_SSS_Fail`: The operation has failed.

## Function `sss_mbedtlsls_associate_ecdhctx`

- Defined in file `_sss_plugin_mbedtlsls_sss_mbedtlsls.h`

### Function Documentation

`int sss_mbedtlsls_associate_ecdhctx (mbedtlsls_ssl_handshake_params *handshake, sss_object_t *pkeyObject, sss_key_store_t *hostKs)`  
Update ECDSA HandShake key with given index.

**Return** 0 if successful, or 1 if unsuccessful

#### Parameters

- `[inout] handshake`: Pointer to the `mbedtlsls_ssl_handshake_params` which will be associated with data corresponding to the key index
- `[in] pkeyObject`: The object that we are going to be use.
- `[in] hostKs`: Keystore to host for session key.

## Function `sss_mbedtls_associate_keypair`

- Defined in file `sss_plugin_mbedtls_sss_mbedtls.h`

### Function Documentation

int **sss\_mbedtls\_associate\_keypair** (mbedtls\_pk\_context \**pkey*, *sss\_object\_t* \**pkeyObject*)  
Associate a keypair provisioned in the secure element for subsequent operations.

**Return** 0 if successful, or 1 if unsuccessful

#### Parameters

- [out] *pkey*: Pointer to the `mbedtls_pk_context` which will be associated with data corresponding to the `key_index`
- [in] *pkeyObject*: The object that we are going to be use.

## Function `sss_mbedtls_associate_pubkey`

- Defined in file `sss_plugin_mbedtls_sss_mbedtls.h`

### Function Documentation

int **sss\_mbedtls\_associate\_pubkey** (mbedtls\_pk\_context \**pkey*, *sss\_object\_t* \**pkeyObject*)  
Associate a pubkey provisioned in the secure element for subsequent operations.

**Return** 0 if successful, or 1 if unsuccessful

#### Parameters

- [out] *pkey*: Pointer to the `mbedtls_pk_context` which will be associated with data corresponding to the `key_index`
- [in] *pkeyObject*: The object that we are going to be use.

## Function `sss_rng_context_free`

- Defined in file `sss_inc_fsl_sss_api.h`

### Function Documentation

*sss\_status\_t* **sss\_rng\_context\_free** (*sss\_rng\_context\_t* \**context*)  
free random genertor context.

**Return** status

#### Parameters

- *context*: generator context.

## Function `sss_rng_context_init`

- Defined in file `sss_inc_fsl_sss_api.h`

### Function Documentation

*sss\_status\_t* **sss\_rng\_context\_init** (*sss\_rng\_context\_t* \*context, *sss\_session\_t* \*session)

Initialise random generator context between application and a security subsystem.

#### Warning API Changed

```
Earlier:
 sss_status_t sss_rng_context_init(
 sss_session_t *session, sss_rng_context_t *context);

Now: Parameters are swapped
 sss_status_t sss_rng_context_init(
 sss_rng_context_t *context, sss_session_t *session);
```

**Return** status

#### Parameters

- session: Session context.
- context: random generator context.

## Function `sss_rng_get_random`

- Defined in file `sss_inc_fsl_sss_api.h`

### Function Documentation

*sss\_status\_t* **sss\_rng\_get\_random** (*sss\_rng\_context\_t* \*context, *uint8\_t* \*random\_data, *size\_t* dataLen)

Generate random number.

**Return** status

#### Parameters

- context: random generator context.
- random\_data: buffer to hold random data.
- dataLen: required random number length

## Function `sss_se05x_aead_context_free`

- Defined in file `sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

void **sss\_se05x\_aead\_context\_free** (*sss\_se05x\_aead\_t* \*context)  
 AEAD context release. The function frees aead context.

#### Parameters

- context: Pointer to aead context.

## Function `sss_se05x_aead_context_init`

- Defined in file `sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

*sss\_status\_t* **sss\_se05x\_aead\_context\_init** (*sss\_se05x\_aead\_t* \*context, *sss\_se05x\_session\_t* \*session, *sss\_se05x\_object\_t* \*keyObject, *sss\_algorithm\_t* algorithm, *sss\_mode\_t* mode)  
 AEAD context init. The function initializes aead context with initial values.

**Return** Status of the operation

#### Parameters

- context: Pointer to aead crypto context.
- session: Associate SSS session with aead context.
- keyObject: Associate SSS key object with aead context.
- algorithm: One of the aead algorithms defined by *sss\_algorithm\_t*.
- mode: One of the modes defined by *sss\_mode\_t*.

#### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.
- *kStatus\_SSS\_InvalidArgument*: One of the arguments is invalid for the function to execute.

## Function `sss_se05x_aead_finish`

- Defined in file `_sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

*sss\_status\_t* **sss\_se05x\_aead\_finish** (*sss\_se05x\_aead\_t* \*context, const uint8\_t \*srcData, size\_t srcLen, uint8\_t \*destData, size\_t destLen, uint8\_t \*tag, size\_t tagLen)

Finalize AEAD. The functions processes data that has not been processed by previous calls to *sss\_aead\_update()* as well as srcData. It finalizes the AEAD operations and computes the tag (encryption) or compares the computed tag with the tag supplied in the parameter (decryption).

**Return** Status of the operation

#### Parameters

- context: Pointer to aead crypto context.
- srcData: Buffer containing final chunk of input data.
- srcLen: Length of final chunk of input data in bytes.
- destData: Buffer containing output data.
- [inout] destLen: Length of output data in bytes. Buffer length on entry, reflects actual output size on return.
- tag: Encryption: Output buffer filled with computed tag Decryption: Input buffer filled with received tag
- tagLen: Length of the computed or received tag in bytes. For AES-GCM it must be 4,8,12,13,14,15 or 16. For AES-CCM it must be 4,6,8,10,12,14 or 16.

#### Return Value

- kStatus\_SSS\_Success*: The operation has completed successfully.
- kStatus\_SSS\_Fail*: The operation has failed.
- kStatus\_SSS\_InvalidArgument*: One of the arguments is invalid for the function to execute.

## Function `sss_se05x_aead_init`

- Defined in file `_sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

*sss\_status\_t* **sss\_se05x\_aead\_init** (*sss\_se05x\_aead\_t* \*context, uint8\_t \*nonce, size\_t nonceLen, size\_t tagLen, size\_t aadLen, size\_t payloadLen)

AEAD init. The function starts the aead operation.

**Return** Status of the operation

#### Parameters

- context: Pointer to aead crypto context.

- `nonce`: The operation nonce or IV.
- `nonceLen`: The length of nonce in bytes. For AES-GCM it must be  $\geq 1$ . For AES-CCM it must be 7, 8, 9, 10, 11, 12, or 13.
- `tagLen`: Length of the computed or received tag in bytes. For AES-GCM it must be 4,8,12,13,14,15 or 16. For AES-CCM it must be 4,6,8,10,12,14 or 16.
- `aadLen`: Input size in bytes of AAD. Used only for AES-CCM. Ignored for AES-GCM.
- `payloadLen`: Length in bytes of the payload. Used only for AES-CCM. Ignored for AES-GCM.

#### Return Value

- `kStatus_SSS_Success`: The operation has completed successfully.
- `kStatus_SSS_Fail`: The operation has failed.

#### Function `sss_se05x_aead_one_go`

- Defined in file `_sss_inc_fsl_sss_se05x_apis.h`

#### Function Documentation

```
sss_status_t sss_se05x_aead_one_go (sss_se05x_aead_t *context, const uint8_t *srcData, uint8_t
 *destData, size_t size, uint8_t *nonce, size_t nonceLen, const
 uint8_t *aad, size_t aadLen, uint8_t *tag, size_t *tagLen)
```

AEAD in one blocking function call. The function blocks current thread until the operation completes or an error occurs.

**Return** Status of the operation

#### Parameters

- `context`: Pointer to aead crypto context.
- `srcData`: Buffer containing the input data.
- `destData`: Buffer containing the output data.
- `size`: Size of input and output data buffer in bytes.
- `nonce`: The operation nonce or IV.
- `nonceLen`: The length of nonce in bytes. For AES-GCM it must be  $\geq 1$ . For AES-CCM it must be 7, 8, 9, 10, 11, 12, or 13.
- `aad`: Input additional authentication data AAD
- `aadLen`: Input size in bytes of AAD
- `tag`: Encryption: Output buffer filled with computed tag Decryption: Input buffer filled with received tag
- `tagLen`: Length of the tag in bytes. For AES-GCM it must be 4,8,12,13,14,15 or 16. For AES-CCM it must be 4,6,8,10,12,14 or 16.

#### Return Value

- `kStatus_SSS_Success`: The operation has completed successfully.
- `kStatus_SSS_Fail`: The operation has failed.

## Function `sss_se05x_aead_update`

- Defined in file `_sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

*sss\_status\_t* **sss\_se05x\_aead\_update** (*sss\_se05x\_aead\_t* \*context, **const** uint8\_t \*srcData, size\_t srcLen, uint8\_t \*destData, size\_t \*destLen)

AEAD data update. Feeds a new chunk of the data payload. Input data does not have to be a multiple of block size. Subsequent calls to this function are possible. Unless one or more calls of this function have supplied sufficient input data, no output is generated. The integration check is done by `sss_aead_finish()`. Until then it is not sure if the decrypt data is authentic.

**Return** Status of the operation

#### Parameters

- context: Pointer to aead crypto context.
- srcData: Buffer containing the input data.
- srcLen: Length of the input data in bytes.
- destData: Buffer containing the output data.
- [inout] destLen: Length of the output data in bytes. Buffer length on entry, reflects actual output size on return.

#### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.
- *kStatus\_SSS\_InvalidArgument*: One of the arguments is invalid for the function to execute.

## Function `sss_se05x_aead_update_aad`

- Defined in file `_sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

*sss\_status\_t* **sss\_se05x\_aead\_update\_aad** (*sss\_se05x\_aead\_t* \*context, **const** uint8\_t \*aadData, size\_t aadDataLen)

Feeds a new chunk of the AAD. Subsequent calls of this function are possible.

**Return** Status of the operation

#### Parameters

- context: Pointer to aead crypto context
- aadData: Input buffer containing the chunk of AAD
- aadDataLen: Length of the AAD data in bytes.

#### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.

- *kStatus\_SSS\_Fail*: The operation has failed.
- *kStatus\_SSS\_InvalidArgument*: One of the arguments is invalid for the function to execute.

## Function `sss_se05x_asymmetric_context_free`

- Defined in file `_sss_inc_fsl_sss_se05x_apis.h`

## Function Documentation

void **sss\_se05x\_asymmetric\_context\_free** (*sss\_se05x\_asymmetric\_t* \*context)  
 Asymmetric context release. The function frees asymmetric context.

### Parameters

- context: Pointer to asymmetric context.

## Function `sss_se05x_asymmetric_context_init`

- Defined in file `_sss_inc_fsl_sss_se05x_apis.h`

## Function Documentation

*sss\_status\_t* **sss\_se05x\_asymmetric\_context\_init** (*sss\_se05x\_asymmetric\_t* \*context, *sss\_se05x\_session\_t* \*session, *sss\_se05x\_object\_t* \*keyObject, *sss\_algorithm\_t* algorithm, *sss\_mode\_t* mode)

Asymmetric context init. The function initializes asymmetric context with initial values.

**Return** Status of the operation

### Parameters

- context: Pointer to asymmetric crypto context.
- session: Associate SSS session with asymmetric context.
- keyObject: Associate SSS key object with asymmetric context.
- algorithm: One of the asymmetric algorithms defined by *sss\_algorithm\_t*.
- mode: One of the modes defined by *sss\_mode\_t*.

### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.
- *kStatus\_SSS\_InvalidArgument*: One of the arguments is invalid for the function to execute.



## Function `sss_se05x_asymmetric_decrypt`

- Defined in file `sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

*sss\_status\_t* **sss\_se05x\_asymmetric\_decrypt** (*sss\_se05x\_asymmetric\_t* \*context, **const** uint8\_t \*srcData, size\_t srcLen, uint8\_t \*destData, size\_t \*destLen)

Asymmetric decryption The function uses asymmetric algorithm to decrypt data. Private key portion of a key pair is used for decryption.

**Return** Status of the operation

#### Parameters

- context: Pointer to asymmetric context.
- srcData: Input buffer
- srcLen: Length of the input in bytes
- destData: Output buffer
- destLen: Length of the output in bytes

#### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.
- *kStatus\_SSS\_InvalidArgument*: One of the arguments is invalid for the function to execute.

## Function `sss_se05x_asymmetric_encrypt`

- Defined in file `sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

*sss\_status\_t* **sss\_se05x\_asymmetric\_encrypt** (*sss\_se05x\_asymmetric\_t* \*context, **const** uint8\_t \*srcData, size\_t srcLen, uint8\_t \*destData, size\_t \*destLen)

Asymmetric encryption The function uses asymmetric algorithm to encrypt data. Public key portion of a key pair is used for encryption.

**Return** Status of the operation

#### Parameters

- context: Pointer to asymmetric context.
- srcData: Input buffer
- srcLen: Length of the input in bytes
- destData: Output buffer
- destLen: Length of the output in bytes

### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.
- *kStatus\_SSS\_InvalidArgument*: One of the arguments is invalid for the function to execute.

### Function `sss_se05x_asymmetric_sign`

- Defined in file `_sss_inc_fsl_sss_se05x_types.h`

### Function Documentation

*sss\_status\_t* **sss\_se05x\_asymmetric\_sign** (*sss\_se05x\_asymmetric\_t* \*context, uint8\_t \*srcData, size\_t srcLen, uint8\_t \*signature, size\_t \*signatureLen)

Similar to *sss\_se05x\_asymmetric\_sign\_digest*,  
but hashing/digest done by SE

### Function `sss_se05x_asymmetric_sign_digest`

- Defined in file `_sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

*sss\_status\_t* **sss\_se05x\_asymmetric\_sign\_digest** (*sss\_se05x\_asymmetric\_t* \*context, uint8\_t \*digest, size\_t digestLen, uint8\_t \*signature, size\_t \*signatureLen)

Asymmetric signature of a message digest The function signs a message digest.

**Return** Status of the operation

### Parameters

- context: Pointer to asymmetric context.
- digest: Input buffer containing the input message digest
- digestLen: Length of the digest in bytes
- signature: Output buffer written with the signature of the digest
- signatureLen: Length of the signature in bytes

### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.
- *kStatus\_SSS\_InvalidArgument*: One of the arguments is invalid for the function to execute.

## Function `sss_se05x_asymmetric_verify`

- Defined in file `sss_inc_fsl_sss_se05x_types.h`

### Function Documentation

`sss_status_t sss_se05x_asymmetric_verify` (`sss_se05x_asymmetric_t *context`, `uint8_t *srcData`, `size_t srcLen`, `uint8_t *signature`, `size_t signatureLen`)  
 Similar to `sss_se05x_asymmetric_verify_digest`, but hashing/digest done by SE

## Function `sss_se05x_asymmetric_verify_digest`

- Defined in file `sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

`sss_status_t sss_se05x_asymmetric_verify_digest` (`sss_se05x_asymmetric_t *context`, `uint8_t *digest`, `size_t digestLen`, `uint8_t *signature`, `size_t signatureLen`)

Asymmetric verify of a message digest The function verifies a message digest.

**Return** Status of the operation

#### Parameters

- `context`: Pointer to asymmetric context.
- `digest`: Input buffer containing the input message digest
- `digestLen`: Length of the digest in bytes
- `signature`: Input buffer containing the signature to verify
- `signatureLen`: Length of the signature in bytes

#### Return Value

- `kStatus_SSS_Success`: The operation has completed successfully.
- `kStatus_SSS_Fail`: The operation has failed.
- `kStatus_SSS_InvalidArgument`: One of the arguments is invalid for the function to execute.

## Function `sss_se05x_cipher_crypt_ctr`

- Defined in file `sss_inc_fsl_sss_se05x_apis.h`

## Function Documentation

*sss\_status\_t* **sss\_se05x\_cipher\_crypt\_ctr** (*sss\_se05x\_symmetric\_t* \*context, **const** uint8\_t \*srcData, uint8\_t \*destData, size\_t size, uint8\_t \*initialCounter, uint8\_t \*lastEncryptedCounter, size\_t \*szLeft)

Symmetric AES in Counter mode in one blocking function call. The function blocks current thread until the operation completes or an error occurs.

**Return** Status of the operation

### Parameters

- context: Pointer to symmetric crypto context.
- srcData: Buffer containing the input data.
- destData: Buffer containing the output data.
- size: Size of source and destination data buffers in bytes.
- [inout] initialCounter: Input counter (updates on return)
- [out] lastEncryptedCounter: Output cipher of last counter, for chained CTR calls. NULL can be passed if chained calls are not used.
- [out] szLeft: Output number of bytes in left unused in lastEncryptedCounter block. NULL can be passed if chained calls are not used.

### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.

## Function **sss\_se05x\_cipher\_finish**

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

## Function Documentation

*sss\_status\_t* **sss\_se05x\_cipher\_finish** (*sss\_se05x\_symmetric\_t* \*context, **const** uint8\_t \*srcData, size\_t srcLen, uint8\_t \*destData, size\_t \*destLen)

Symmetric cipher finalize.

**Return** Status of the operation

### Parameters

- context: Pointer to symmetric crypto context.
- srcData: Buffer containing final chunk of input data.
- srcLen: Length of final chunk of input data in bytes.
- destData: Buffer containing output data.
- [inout] destLen: Length of output data in bytes. Buffer length on entry, reflects actual output size on return.

### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.
- *kStatus\_SSS\_InvalidArgument*: One of the arguments is invalid for the function to execute.

### Function `sss_se05x_cipher_init`

- Defined in `file_sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

*sss\_status\_t* **sss\_se05x\_cipher\_init** (*sss\_se05x\_symmetric\_t* \*context, uint8\_t \*iv, size\_t ivLen)

Symmetric cipher init. The function starts the symmetric cipher operation.

**Return** Status of the operation

#### Parameters

- context: Pointer to symmetric crypto context.
- iv: Buffer containing the symmetric operation Initialization Vector.
- ivLen: Length of the Initialization Vector in bytes.

#### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.

### Function `sss_se05x_cipher_one_go`

- Defined in `file_sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

*sss\_status\_t* **sss\_se05x\_cipher\_one\_go** (*sss\_se05x\_symmetric\_t* \*context, uint8\_t \*iv, size\_t ivLen, **const** uint8\_t \*srcData, uint8\_t \*destData, size\_t dataLen)

Symmetric cipher in one blocking function call. The function blocks current thread until the operation completes or an error occurs.

**Return** Status of the operation

#### Parameters

- context: Pointer to symmetric crypto context.
- iv: Buffer containing the symmetric operation Initialization Vector.
- ivLen: Length of the Initialization Vector in bytes.
- srcData: Buffer containing the input data.
- destData: Buffer containing the output data.
- dataLen: Size of input and output data buffer in bytes.

#### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.

## Function `sss_se05x_cipher_update`

- Defined in file `_sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

*sss\_status\_t* **sss\_se05x\_cipher\_update** (*sss\_se05x\_symmetric\_t* \*context, **const** uint8\_t \*srcData, size\_t srcLen, uint8\_t \*destData, size\_t \*destLen)

Symmetric cipher update. Input data does not have to be a multiple of block size. Subsequent calls to this function are possible. Unless one or more calls of this function have supplied sufficient input data, no output is generated. The cipher operation is finalized with a call to *sss\_cipher\_finish()*.

**Return** Status of the operation

#### Parameters

- context: Pointer to symmetric crypto context.
- srcData: Buffer containing the input data.
- srcLen: Length of the input data in bytes.
- destData: Buffer containing the output data.
- [inout] destLen: Length of the output data in bytes. Buffer length on entry, reflects actual output size on return.

#### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.
- *kStatus\_SSS\_InvalidArgument*: One of the arguments is invalid for the function to execute.

## Function `sss_se05x_derive_key_context_free`

- Defined in file `_sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

void **sss\_se05x\_derive\_key\_context\_free** (*sss\_se05x\_derive\_key\_t* \*context)

Derive key context release. The function frees derive key context.

#### Parameters

- context: Pointer to derive key context.

## Function `sss_se05x_derive_key_context_init`

- Defined in file `_sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

```
sss_status_t sss_se05x_derive_key_context_init (sss_se05x_derive_key_t *con-
 text, sss_se05x_session_t *ses-
 sion, sss_se05x_object_t *keyObject,
 sss_algorithm_t algorithm, sss_mode_t mode)
```

Derive key context init. The function initializes derive key context with initial values.

**Return** Status of the operation

#### Parameters

- `context`: Pointer to derive key context.
- `session`: Associate SSS session with the derive key context.
- `keyObject`: Associate SSS key object with the derive key context.
- `algorithm`: One of the derive key algorithms defined by *sss\_algorithm\_t*.
- `mode`: One of the modes defined by *sss\_mode\_t*.

#### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.
- *kStatus\_SSS\_InvalidArgument*: One of the arguments is invalid for the function to execute.

## Function `sss_se05x_derive_key_dh`

- Defined in file `_sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

```
sss_status_t sss_se05x_derive_key_dh (sss_se05x_derive_key_t *context, sss_se05x_object_t *other-
 PartyKeyObject, sss_se05x_object_t *derivedKeyObject)
```

Asymmetric key derivation Diffie-Hellmann The function cryptographically derives a key from another key. For example Diffie-Hellmann.

**Return** Status of the operation

#### Parameters

- `context`: Pointer to derive key context.
- `otherPartyKeyObject`: Public key of the other party in the Diffie-Hellmann algorithm
- `[inout] derivedKeyObject`: Reference to a derived key

#### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.

- *kStatus\_SSS\_Fail*: The operation has failed.
- *kStatus\_SSS\_InvalidArgument*: One of the arguments is invalid for the function to execute.

## Function `sss_se05x_derive_key_go`

- Defined in file `_sss_inc_fsl_sss_se05x_apis.h`

## Function Documentation

```
sss_status_t sss_se05x_derive_key_go(sss_se05x_derive_key_t *context, const uint8_t *saltData, size_t saltLen, const uint8_t *info, size_t infoLen, sss_se05x_object_t *derivedKeyObject, uint16_t deriveDataLen, uint8_t *hkdfOutput, size_t *hkdfOutputLen)
```

Symmetric key derivation The function cryptographically derives a key from another key. For example MIFARE key derivation, PRF, HKDF-Extract.

**Return** Status of the operation

### Parameters

- context: Pointer to derive key context.
- saltData: Input data buffer, typically with some random data.
- saltLen: Length of saltData buffer in bytes.
- info: Input data buffer, typically with some fixed info.
- infoLen: Length of info buffer in bytes.
- [inout] derivedKeyObject: Reference to a derived key
- deriveDataLen: **TODO** Document this
- hkdfOutput: **TODO** Document this
- hkdfOutputLen: **TODO** Document this

### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.
- *kStatus\_SSS\_InvalidArgument*: One of the arguments is invalid for the function to execute.

## Function `sss_se05x_derive_key_one_go`

- Defined in file `_sss_inc_fsl_sss_se05x_apis.h`



## Function Documentation

*sss\_status\_t* **sss\_se05x\_derive\_key\_one\_go** (*sss\_se05x\_derive\_key\_t* \*context, **const** uint8\_t \*saltData, size\_t saltLen, **const** uint8\_t \*info, size\_t infoLen, *sss\_se05x\_object\_t* \*derivedKeyObject, uint16\_t deriveDataLen)

Symmetric key derivation (replaces the deprecated function *sss\_derive\_key\_go*) The function cryptographically derives a key from another key. For example MIFARE key derivation, PRF, HKDF-Extract-Expand, HKDF-Expand. Refer to *sss\_derive\_key\_sobj\_one\_go* in case the Salt is available as a key object.

**Return** Status of the operation

### Parameters

- context: Pointer to derive key context.
- saltData: Input data buffer, typically with some random data.
- saltLen: Length of saltData buffer in bytes.
- info: Input data buffer, typically with some fixed info.
- infoLen: Length of info buffer in bytes.
- [inout] derivedKeyObject: Reference to a derived key
- [in] deriveDataLen: Expected length of derived key.

### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.
- *kStatus\_SSS\_InvalidArgument*: One of the arguments is invalid for the function to execute.

## Function *sss\_se05x\_derive\_key\_sobj\_one\_go*

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

## Function Documentation

*sss\_status\_t* **sss\_se05x\_derive\_key\_sobj\_one\_go** (*sss\_se05x\_derive\_key\_t* \*context, *sss\_se05x\_object\_t* \*saltKeyObject, **const** uint8\_t \*info, size\_t infoLen, *sss\_se05x\_object\_t* \*derivedKeyObject, uint16\_t deriveDataLen)

Symmetric key derivation (salt in key object) Refer to *sss\_derive\_key\_one\_go* in case the salt is not available as a key object.

**Return** Status of the operation

### Parameters

- context: Pointer to derive key context
- saltKeyObject: Reference to salt. The salt key object must reside in the same keystore as the derive key context.
- [in] info: Input data buffer, typically with some fixed info.
- [in] infoLen: Length of info buffer in bytes.

- `derivedKeyObject`: Reference to a derived key
- `[in] deriveDataLen`: The derive data length

#### Return Value

- `kStatus_SSS_Success`: The operation has completed successfully.
- `kStatus_SSS_Fail`: The operation has failed.
- `kStatus_SSS_InvalidArgument`: One of the arguments is invalid for the function to execute.

### Function `sss_se05x_digest_context_free`

- Defined in `file_sss_inc_fsl_sss_se05x_apis.h`

#### Function Documentation

void **`sss_se05x_digest_context_free`** (*`sss_se05x_digest_t` \*context*)  
Digest context release. The function frees digest context.

#### Parameters

- `context`: Pointer to digest context.

### Function `sss_se05x_digest_context_init`

- Defined in `file_sss_inc_fsl_sss_se05x_apis.h`

#### Function Documentation

*`sss_status_t`* **`sss_se05x_digest_context_init`** (*`sss_se05x_digest_t` \*context, `sss_se05x_session_t` \*session, `sss_algorithm_t` algorithm, `sss_mode_t` mode*)  
Digest context init. The function initializes digest context with initial values.

**Return** Status of the operation

#### Parameters

- `context`: Pointer to digest context.
- `session`: Associate SSS session with digest context.
- `algorithm`: One of the digest algorithms defined by *`sss_algorithm_t`*.
- `mode`: One of the modes defined by *`sss_mode_t`*.

#### Return Value

- `kStatus_SSS_Success`: The operation has completed successfully.
- `kStatus_SSS_Fail`: The operation has failed.
- `kStatus_SSS_InvalidArgument`: One of the arguments is invalid for the function to execute.

## Function `sss_se05x_digest_finish`

- Defined in file `_sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

*sss\_status\_t* **sss\_se05x\_digest\_finish** (*sss\_se05x\_digest\_t* \*context, uint8\_t \*digest, size\_t \*digestLen)

Finish digest for a message. The function blocks current thread until the operation completes or an error occurs.

**Return** Status of the operation

#### Parameters

- context: Pointer to digest context.
- digest: Output message digest
- digestLen: Message digest byte length

#### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.

## Function `sss_se05x_digest_init`

- Defined in file `_sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

*sss\_status\_t* **sss\_se05x\_digest\_init** (*sss\_se05x\_digest\_t* \*context)

Init digest for a message. The function blocks current thread until the operation completes or an error occurs.

**Return** Status of the operation

#### Parameters

- context: Pointer to digest context.

#### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.

## Function `sss_se05x_digest_one_go`

- Defined in file `_sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

*sss\_status\_t* **sss\_se05x\_digest\_one\_go** (*sss\_se05x\_digest\_t* \*context, **const** uint8\_t \*message, size\_t messageLen, uint8\_t \*digest, size\_t \*digestLen)

Message digest in one blocking function call. The function blocks current thread until the operation completes or an error occurs.

**Return** Status of the operation

#### Parameters

- context: Pointer to digest context.
- message: Input message
- messageLen: Length of the input message in bytes
- digest: Output message digest
- digestLen: Message digest byte length

#### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.

## Function `sss_se05x_digest_update`

- Defined in file `_sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

*sss\_status\_t* **sss\_se05x\_digest\_update** (*sss\_se05x\_digest\_t* \*context, **const** uint8\_t \*message, size\_t messageLen)

Update digest for a message.

The function blocks current thread until the operation completes or an error occurs.

**Return** Status of the operation

#### Parameters

- context: Pointer to digest context.
- message: Buffer with a message chunk.
- messageLen: Length of the input buffer in bytes.

#### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.

## Function `sss_se05x_key_object_allocate_handle`

- Defined in file `sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

```
sss_status_t sss_se05x_key_object_allocate_handle (sss_se05x_object_t *keyObject,
 uint32_t keyId, sss_key_part_t key-
 Part, sss_cipher_type_t cipherType, size_t
 keyByteLenMax, uint32_t options)
```

Allocate / pre-provision memory for new key.

This API allows underlying cryptographic subsystems to perform preconditions of before creating any cryptographic key object.

On SE050, the memory get reserved only when the actual object is created and hence there is no memory reservation happening in this API call. but internally it checks if the object already exists or not . if the object is already existing it returns a failure.

**Return** Status of object allocation.

#### Parameters

- [inout] `keyObject`: The object If required, update implementation defined values inside the `keyObject`
- `keyId`: External Key ID. Later on this may be used by `sss_key_object_get_handle`
- `keyPart`: See `sss_key_part_t`
- `cipherType`: See `sss_cipher_type_t`
- `keyByteLenMax`: Maximum storage this type of key may need. For systems that have their own internal allocation table this would help
- `options`: 0 = Persistent Key (Default) or Transient Key. See `sss_key_object_mode_t`

## Function `sss_se05x_key_object_free`

- Defined in file `sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

```
void sss_se05x_key_object_free (sss_se05x_object_t *keyObject)
```

Destructor for the key object. The function frees key object context.

On SE050, this has no impact on physical Key Object.

#### Parameters

- `keyObject`: Pointer to key object context.

## Function `sss_se05x_key_object_get_access`

- Defined in file `_sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

*sss\_status\_t* **sss\_se05x\_key\_object\_get\_access** (*sss\_se05x\_object\_t* \*keyObject, uint32\_t \*access)  
Not Available for SE05X

## Function `sss_se05x_key_object_get_handle`

- Defined in file `_sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

*sss\_status\_t* **sss\_se05x\_key\_object\_get\_handle** (*sss\_se05x\_object\_t* \*keyObject, uint32\_t keyId)  
Get handle to an existing allocated/provisioned/created Object.

See *sss\_key\_object\_allocate\_handle*.

After calling this API, Ideally keyObject should become equivalent to as set after the calling of *sss\_key\_object\_allocate\_handle* api.

On SE05X, this API uses *Se05x\_API\_ReadType* and fetches parameters of the API.

**Return** The sss status.

#### Parameters

- keyObject: The key object
- [in] keyId: The key identifier

## Function `sss_se05x_key_object_get_purpose`

- Defined in file `_sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

*sss\_status\_t* **sss\_se05x\_key\_object\_get\_purpose** (*sss\_se05x\_object\_t* \*keyObject, *sss\_mode\_t* \*purpose)  
Not Available for SE05X

## Function `sss_se05x_key_object_get_user`

- Defined in file `sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

*sss\_status\_t* **sss\_se05x\_key\_object\_get\_user** (*sss\_se05x\_object\_t* \*keyObject, uint32\_t \*user)  
Not Available for SE05X

## Function `sss_se05x_key_object_init`

- Defined in file `sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

*sss\_status\_t* **sss\_se05x\_key\_object\_init** (*sss\_se05x\_object\_t* \*keyObject, *sss\_se05x\_key\_store\_t* \*keyStore)

Constructor for a key object data structure The function initializes keyObject data structure and associates it with a key store in which the plain key and other attributes are stored.

**Return** Status of the operation

#### Parameters

- keyObject:
- keyStore:

#### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.
- *kStatus\_SSS\_InvalidArgument*: One of the arguments is invalid for the function to execute.

## Function `sss_se05x_key_object_set_access`

- Defined in file `sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

*sss\_status\_t* **sss\_se05x\_key\_object\_set\_access** (*sss\_se05x\_object\_t* \*keyObject, uint32\_t access, uint32\_t options)

Not Available for SE05X

## Function `sss_se05x_key_object_set_eccgfp_group`

- Defined in file `_sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

*sss\_status\_t* **sss\_se05x\_key\_object\_set\_eccgfp\_group** (*sss\_se05x\_object\_t* \*keyObject,  
*sss\_eccgfp\_group\_t* \*group)

Not Available for SE05X

## Function `sss_se05x_key_object_set_purpose`

- Defined in file `_sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

*sss\_status\_t* **sss\_se05x\_key\_object\_set\_purpose** (*sss\_se05x\_object\_t* \*keyObject, *sss\_mode\_t* purpose,  
uint32\_t options)

Assign purpose to a key object.

#### Parameters

- keyObject: the object where permission restrictions are applied
- purpose: Usage of the key.
- options: Transient or persistent update. Allows for transient update of persistent attributes.

## Function `sss_se05x_key_object_set_user`

- Defined in file `_sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

*sss\_status\_t* **sss\_se05x\_key\_object\_set\_user** (*sss\_se05x\_object\_t* \*keyObject, uint32\_t user,  
uint32\_t options)

Not Available for SE05X

## Function `sss_se05x_key_store_allocate`

- Defined in file `_sss_inc_fsl_sss_se05x_apis.h`



## Function Documentation

*sss\_status\_t* **sss\_se05x\_key\_store\_allocate** (*sss\_se05x\_key\_store\_t* \*keyStore, uint32\_t keyStoreId)

Get handle to key store. If the key store already exists, nothing is allocated. If the key store does not exist, new empty key store is created and initialized. Key store context structure is updated with actual information.

This API does not do anything special on SE05X.

### Parameters

- [out] keyStore: Pointer to key store context. Key store context is updated on function return.
- keyStoreId: Implementation specific ID, can be used in case security subsystem manages multiple different key stores.

## Function **sss\_se05x\_key\_store\_context\_free**

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

## Function Documentation

void **sss\_se05x\_key\_store\_context\_free** (*sss\_se05x\_key\_store\_t* \*keyStore)

Destructor for the key store context.

## Function **sss\_se05x\_key\_store\_context\_init**

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

## Function Documentation

*sss\_status\_t* **sss\_se05x\_key\_store\_context\_init** (*sss\_se05x\_key\_store\_t* \*keyStore, *sss\_se05x\_session\_t* \*session)

Constructor for the key store context data structure.

### Parameters

- [out] keyStore: Pointer to key store context. Key store context is updated on function return.
- session: Session context.

## Function **sss\_se05x\_key\_store\_create\_curve**

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

## Function Documentation

*sss\_status\_t* **sss\_se05x\_key\_store\_create\_curve** (*Se05xSession\_t \*pSession*, *uint32\_t curve\_id*)

## Function sss\_se05x\_key\_store\_erase\_key

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

## Function Documentation

*sss\_status\_t* **sss\_se05x\_key\_store\_erase\_key** (*sss\_se05x\_key\_store\_t \*keyStore*, *sss\_se05x\_object\_t \*keyObject*)

Delete / destroy allocated keyObect .

**Return** The sss status.

### Parameters

- *keyStore*: The key store
- *keyObject*: The key object to be deleted

## Function sss\_se05x\_key\_store\_export\_key

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

## Function Documentation

*sss\_status\_t* **sss\_se05x\_key\_store\_export\_key** (*sss\_se05x\_key\_store\_t \*keyStore*, *sss\_se05x\_object\_t \*keyObject*, *uint8\_t \*key*, *size\_t \*keylen*)

Export Key from SE050 to host

Only Transient keys can be exported.

## Function sss\_se05x\_key\_store\_freeze\_key

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

## Function Documentation

*sss\_status\_t* **sss\_se05x\_key\_store\_freeze\_key** (*sss\_se05x\_key\_store\_t \*keyStore*, *sss\_se05x\_object\_t \*keyObject*)

Not available for SE05X

## Function `sss_se05x_key_store_generate_key`

- Defined in file `sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

```
sss_status_t sss_se05x_key_store_generate_key (sss_se05x_key_store_t *keyStore,
sss_se05x_object_t *keyObject, size_t key-
BitLen, void *options)
```

This function generates key[] in the destination key store.

## Function `sss_se05x_key_store_get_key`

- Defined in file `sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

```
sss_status_t sss_se05x_key_store_get_key (sss_se05x_key_store_t *keyStore, sss_se05x_object_t
*keyObject, uint8_t *data, size_t *dataLen, size_t
*pKeyBitLen)
```

This function exports plain key[] from key store (if constraints and user id allows reading)

## Function `sss_se05x_key_store_get_key_attst`

- Defined in file `sss_inc_fsl_sss_se05x_types.h`

### Function Documentation

```
sss_status_t sss_se05x_key_store_get_key_attst (sss_se05x_key_store_t *keyStore,
sss_se05x_object_t *keyObject, uint8_t
*key, size_t *keylen, size_t *pKeyBitLen,
sss_se05x_object_t *keyObject_attst,
sss_algorithm_t algorithm_attst, uint8_t
*random_attst, size_t randomLen_attst,
sss_se05x_attst_data_t *attst_data)
```

Read with attestation

## Function `sss_se05x_key_store_import_key`

- Defined in file `sss_inc_fsl_sss_se05x_apis.h`

## Function Documentation

*sss\_status\_t* **sss\_se05x\_key\_store\_import\_key** (*sss\_se05x\_key\_store\_t* \*keyStore,  
*sss\_se05x\_object\_t* \*keyObject, uint8\_t \*key,  
 size\_t keylen)

Re Import previously exported SE05X key from host to the SE05X

Only Transient keys can be imported.

## Function sss\_se05x\_key\_store\_load

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

## Function Documentation

*sss\_status\_t* **sss\_se05x\_key\_store\_load** (*sss\_se05x\_key\_store\_t* \*keyStore)

Load from persistent memory to cached objects.

This API does not do anything special on SE05X.

## Function sss\_se05x\_key\_store\_open\_key

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

## Function Documentation

*sss\_status\_t* **sss\_se05x\_key\_store\_open\_key** (*sss\_se05x\_key\_store\_t* \*keyStore, *sss\_se05x\_object\_t* \*keyObject)

Access key store using one more level of encryption.

e.g. Access keys / encryption key during storage

In SE05X, these keys can be used as KEK encryption key

**Return** The sss status.

### Parameters

- keyStore: The key store
- keyObject: The key object that is to be used as a KEK (Key Encryption Key)

If keyObject == NULL, then subsequent key injection does not use any KEK.

**Return** The sss status.

## Function `sss_se05x_key_store_save`

- Defined in file `sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

*sss\_status\_t* **sss\_se05x\_key\_store\_save** (*sss\_se05x\_key\_store\_t* \*keyStore)

Save all cached persistent objects to persistent memory.

This API does not do anything special on SE05X.

## Function `sss_se05x_key_store_set_key`

- Defined in file `sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

*sss\_status\_t* **sss\_se05x\_key\_store\_set\_key** (*sss\_se05x\_key\_store\_t* \*keyStore, *sss\_se05x\_object\_t* \*keyObject, **const** *uint8\_t* \*data, *size\_t* dataLen, *size\_t* keyBitLen, *void* \*options, *size\_t* optionsLen)

This function moves data[] from memory to the destination key store.

#### Return

#### Parameters

- keyStore: Key store context
- keyObject: Reference to a key and it's properties
- data: Data to be stored in Key. When setting ecc private key only, do not include key header.
- dataLen: Length of the data
- keyBitLen: Crypto algorithm key bit length
- options: Pointer to implementation specific options
- optionsLen: Length of the options in bytes

## Function `sss_se05x_mac_context_free`

- Defined in file `sss_inc_fsl_sss_se05x_apis.h`

## Function Documentation

void **sss\_se05x\_mac\_context\_free** (*sss\_se05x\_mac\_t* \*context)  
MAC context release. The function frees mac context.

### Parameters

- context: Pointer to mac context.

## Function **sss\_se05x\_mac\_context\_init**

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

## Function Documentation

*sss\_status\_t* **sss\_se05x\_mac\_context\_init** (*sss\_se05x\_mac\_t* \*context, *sss\_se05x\_session\_t* \*session, *sss\_se05x\_object\_t* \*keyObject, *sss\_algorithm\_t* algorithm, *sss\_mode\_t* mode)  
MAC context init. The function initializes mac context with initial values.

**Return** Status of the operation

### Parameters

- context: Pointer to mac context.
- session: Associate SSS session with mac context.
- keyObject: Associate SSS key object with mac context.
- algorithm: One of the mac algorithms defined by *sss\_algorithm\_t*.
- mode: One of the modes defined by *sss\_mode\_t*.

### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.
- *kStatus\_SSS\_InvalidArgument*: One of the arguments is invalid for the function to execute.

## Function **sss\_se05x\_mac\_finish**

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

## Function Documentation

*sss\_status\_t* **sss\_se05x\_mac\_finish** (*sss\_se05x\_mac\_t* \*context, uint8\_t \*mac, size\_t \*macLen)

Finish mac for a message. The function blocks current thread until the operation completes or an error occurs.

**Return** Status of the operation

### Parameters

- context: Pointer to mac context.
- mac: Output message MAC
- macLen: Computed MAC byte length

### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.

## Function sss\_se05x\_mac\_init

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

## Function Documentation

*sss\_status\_t* **sss\_se05x\_mac\_init** (*sss\_se05x\_mac\_t* \*context)

Init mac for a message. The function blocks current thread until the operation completes or an error occurs.

**Return** Status of the operation

### Parameters

- context: Pointer to mac context.

### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.

## Function sss\_se05x\_mac\_one\_go

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

## Function Documentation

*sss\_status\_t* **sss\_se05x\_mac\_one\_go** (*sss\_se05x\_mac\_t* \*context, const uint8\_t \*message, size\_t messageLen, uint8\_t \*mac, size\_t \*macLen)

Message MAC in one blocking function call. The function blocks current thread until the operation completes or an error occurs.

**Return** Status of the operation

### Parameters

- context: Pointer to mac context.
- message: Input message
- messageLen: Length of the input message in bytes
- mac: Output message MAC
- macLen: Computed MAC byte length

#### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.

### Function sss\_se05x\_mac\_update

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

#### Function Documentation

*sss\_status\_t* **sss\_se05x\_mac\_update** (*sss\_se05x\_mac\_t* \*context, **const** uint8\_t \*message, size\_t messageLen)

Update mac for a message.

The function blocks current thread until the operation completes or an error occurs.

**Return** Status of the operation

#### Parameters

- context: Pointer to mac context.
- message: Buffer with a message chunk.
- messageLen: Length of the input buffer in bytes.

#### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.

### Function sss\_se05x\_mac\_validate\_one\_go

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_types.h

#### Function Documentation

*sss\_status\_t* **sss\_se05x\_mac\_validate\_one\_go** (*sss\_se05x\_mac\_t* \*context, **const** uint8\_t \*message, size\_t messageLen, uint8\_t \*mac, size\_t macLen)

MAC Validate



## Function sss\_se05x\_refresh\_session

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Function Documentation

*sss\_status\_t* **sss\_se05x\_refresh\_session** (*sss\_se05x\_session\_t* \*session, void \*connectionData)

## Function sss\_se05x\_rng\_context\_free

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Function Documentation

*sss\_status\_t* **sss\_se05x\_rng\_context\_free** (*sss\_se05x\_rng\_context\_t* \*context)  
free random genertor context.

**Return** status

#### Parameters

- context: generator context.

## Function sss\_se05x\_rng\_context\_init

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Function Documentation

*sss\_status\_t* **sss\_se05x\_rng\_context\_init** (*sss\_se05x\_rng\_context\_t* \*context, *sss\_se05x\_session\_t* \*session)

Initialise random generator context between application and a security subsystem.

**Warning** API Changed

```
Earlier:
 sss_status_t sss_rng_context_init(
 sss_session_t *session, sss_rng_context_t *context);

Now: Parameters are swapped
 sss_status_t sss_rng_context_init(
 sss_rng_context_t *context, sss_session_t *session);
```

**Return** status

#### Parameters

- session: Session context.
- context: random generator context.

## Function `sss_se05x_rng_get_random`

- Defined in file `_sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

`sss_status_t sss_se05x_rng_get_random(sss_se05x_rng_context_t *context, uint8_t *random_data, size_t dataLen)`

Generate random number.

**Return** status

#### Parameters

- `context`: random generator context.
- `random_data`: buffer to hold random data.
- `dataLen`: required random number length

## Function `sss_se05x_session_close`

- Defined in file `_sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

void `sss_se05x_session_close(sss_se05x_session_t *session)`

Close session between application and security subsystem.

This function closes a session which has been opened with a security subsystem. All commands within the session must have completed before this function can be called. The implementation must do nothing if the input `session` parameter is NULL.

#### Parameters

- `session`: Session context.

## Function `sss_se05x_session_create`

- Defined in file `_sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

`sss_status_t sss_se05x_session_create(sss_se05x_session_t *session, sss_type_t subsystem, uint32_t application_id, sss_connection_type_t connection_type, void *connectionData)`

Same as `sss_session_open` but to support sub systems that explicitly need a create before opening.

For the sake of portability across various sub systems, the applicaiton has to call `sss_session_create` before calling `sss_session_open`.

#### Parameters

- [inout] session: Pointer to session context
- [in] subsystem: See *sss\_session\_open*
- [in] application\_id: See *sss\_session\_open*
- [in] connection\_type: See *sss\_session\_open*
- [in] connectionData: See *sss\_session\_open*

### Function *sss\_se05x\_session\_delete*

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Function Documentation

void **sss\_se05x\_session\_delete** (*sss\_se05x\_session\_t* \*session)  
Counterpart to *sss\_session\_create*

Similar to constraint on *sss\_session\_create*, application may call *sss\_session\_delete* to explicitly release all underlying/used session specific resources of that implementation.

### Function *sss\_se05x\_session\_open*

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Function Documentation

*sss\_status\_t* **sss\_se05x\_session\_open** (*sss\_se05x\_session\_t* \*session, *sss\_type\_t* subsystem, uint32\_t application\_id, *sss\_connection\_type\_t* connection\_type, void \*connectionData)

Open session between application and a security subsystem.

Open virtual session between application (user context) and a security subsystem and function thereof. Pointer to session shall be supplied to all SSS APIs as argument. Low level SSS functions can provide implementation specific behaviour based on the session argument.

**Return** status

#### Parameters

- [inout] session: Session context.
- [in] subsystem: Indicates which security subsystem is selected to be used.
- [in] application\_id: ObjectId/AuthenticationID Connecting to:
  - application\_id == 0 => Super user / Platform user
  - Anything else => Authenticated user
- [in] connection\_type: How are we connecting to the system.
- [inout] connectionData: subsystem specific connection parameters.

## Function `sss_se05x_session_prop_get_au8`

- Defined in file `_sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

*sss\_status\_t* **sss\_se05x\_session\_prop\_get\_au8** (*sss\_se05x\_session\_t* \*session, uint32\_t property, uint8\_t \*pValue, size\_t \*pValueLen)

Get an underlying property of the crypto sub system.

This API is used to get values that are numeric in nature.

Property can be either fixed value that is calculated at compile time and returned directly, or it may involve some access to the underlying system.

#### Return

#### Parameters

- [in] session: Session context
- [in] property: Value that is part of *sss\_session\_prop\_au8\_t*
- [out] pValue: Output buffer array
- [inout] pValueLen: Count of values there are/must be read

## Function `sss_se05x_session_prop_get_u32`

- Defined in file `_sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

*sss\_status\_t* **sss\_se05x\_session\_prop\_get\_u32** (*sss\_se05x\_session\_t* \*session, uint32\_t property, uint32\_t \*pValue)

Get an underlying property of the crypto sub system.

This API is used to get values that are numeric in nature.

Property can be either fixed value that is calculated at compile time and returned directly, or it may involve some access to the underlying system.

For applicable properties see *sss\_session\_prop\_u32\_t*

#### Return

#### Parameters

- [in] session: Session context
- [in] property: Value that is part of *sss\_session\_prop\_u32\_t*
- [out] pValue:

## Function `sss_se05x_set_feature`

- Defined in file `sss_inc_fsl_sss_se05x_apis.h`

## Function Documentation

*sss\_status\_t* **sss\_se05x\_set\_feature** (*sss\_se05x\_session\_t* \*session, *SE05x\_Applet\_Feature\_t* feature)  
 Set features of the Applet.  
 See *Se05x\_API\_SetAppletFeatures*

## Function `sss_se05x_symmetric_context_free`

- Defined in file `sss_inc_fsl_sss_se05x_apis.h`

## Function Documentation

void **sss\_se05x\_symmetric\_context\_free** (*sss\_se05x\_symmetric\_t* \*context)  
 Symmetric context release. The function frees symmetric context.

### Parameters

- context: Pointer to symmetric crypto context.

## Function `sss_se05x_symmetric_context_init`

- Defined in file `sss_inc_fsl_sss_se05x_apis.h`

## Function Documentation

*sss\_status\_t* **sss\_se05x\_symmetric\_context\_init** (*sss\_se05x\_symmetric\_t* \*context, *sss\_se05x\_session\_t* \*session, *sss\_se05x\_object\_t* \*keyObject, *sss\_algorithm\_t* algorithm, *sss\_mode\_t* mode)  
 Symmetric context init. The function initializes symmetric context with initial values.

**Return** Status of the operation

### Parameters

- context: Pointer to symmetric crypto context.
- session: Associate SSS session with symmetric context.
- keyObject: Associate SSS key object with symmetric context.
- algorithm: One of the symmetric algorithms defined by *sss\_algorithm\_t*.
- mode: One of the modes defined by *sss\_mode\_t*.

### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.

- *kStatus\_SSS\_Fail*: The operation has failed.
- *kStatus\_SSS\_InvalidArgument*: One of the arguments is invalid for the function to execute.

### Function `sss_se05x_tunnel`

- Defined in `file_sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

*sss\_status\_t* **sss\_se05x\_tunnel** (*sss\_se05x\_tunnel\_context\_t* \*context, uint8\_t \*data, size\_t dataLen, *sss\_se05x\_object\_t* \*keyObjects, uint32\_t keyObjectCount, uint32\_t tunnelType)

Tunneling

Used for communication via another system.

### Function `sss_se05x_tunnel_context_free`

- Defined in `file_sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

void **sss\_se05x\_tunnel\_context\_free** (*sss\_se05x\_tunnel\_context\_t* \*context)  
Destructor for the tunnelling service context.

#### Parameters

- [out] context: Pointer to tunnel context.

### Function `sss_se05x_tunnel_context_init`

- Defined in `file_sss_inc_fsl_sss_se05x_apis.h`

### Function Documentation

*sss\_status\_t* **sss\_se05x\_tunnel\_context\_init** (*sss\_se05x\_tunnel\_context\_t* \*context, *sss\_se05x\_session\_t* \*session)

Constructor for the tunnelling service context.

Earlier: `sss_status_t sss_tunnel_context_init( sss_session_t *session, sss_tunnel_t *context);`

Now: Parameters are swapped `sss_status_t sss_tunnel_context_init( sss_tunnel_t *context, sss_session_t *session);`

#### Parameters

- [out] context: Pointer to tunnel context. Tunnel context is updated on function return.
- session: Pointer to session this tunnelling service belongs to.

## Function `sss_session_close`

- Defined in `file_sss_inc_fsl_sss_api.h`

## Function Documentation

void **sss\_session\_close** (*sss\_session\_t* \*session)

Close session between application and security subsystem.

This function closes a session which has been opened with a security subsystem. All commands within the session must have completed before this function can be called. The implementation must do nothing if the input `session` parameter is NULL.

### Parameters

- `session`: Session context.

## Function `sss_session_create`

- Defined in `file_sss_inc_fsl_sss_api.h`

## Function Documentation

*sss\_status\_t* **sss\_session\_create** (*sss\_session\_t* \*session, *sss\_type\_t* subsystem, uint32\_t application\_id, *sss\_connection\_type\_t* connection\_type, void \*connectionData)

Same as *sss\_session\_open* but to support sub systems that explicitly need a create before opening.

For the sake of portability across various sub systems, the applicaiton has to call *sss\_session\_create* before calling *sss\_session\_open*.

### Parameters

- [inout] `session`: Pointer to session context
- [in] `subsystem`: See *sss\_session\_open*
- [in] `application_id`: See *sss\_session\_open*
- [in] `connection_type`: See *sss\_session\_open*
- [in] `connectionData`: See *sss\_session\_open*

## Function `sss_session_delete`

- Defined in `file_sss_inc_fsl_sss_api.h`

## Function Documentation

void **sss\_session\_delete** (*sss\_session\_t* \*session)

Counterpart to *sss\_session\_create*

Similar to constraint on *sss\_session\_create*, application may call *sss\_session\_delete* to explicitly release all underlying/used session specific resources of that implementation.

## Function sss\_session\_open

- Defined in file\_sss\_inc\_fsl\_sss\_api.h

## Function Documentation

*sss\_status\_t* **sss\_session\_open** (*sss\_session\_t* \*session, *sss\_type\_t* subsystem, uint32\_t application\_id, *sss\_connection\_type\_t* connection\_type, void \*connectionData)

Open session between application and a security subsystem.

Open virtual session between application (user context) and a security subsystem and function thereof. Pointer to session shall be supplied to all SSS APIs as argument. Low level SSS functions can provide implementation specific behaviour based on the session argument.

**Return** status

### Parameters

- [inout] session: Session context.
- [in] subsystem: Indicates which security subsystem is selected to be used.
- [in] application\_id: ObjectId/AuthenticationID Connecting to:
  - application\_id == 0 => Super user / Platform user
  - Anything else => Authenticated user
- [in] connection\_type: How are we connecting to the system.
- [inout] connectionData: subsystem specific connection parameters.

## Function sss\_session\_prop\_get\_au8

- Defined in file\_sss\_inc\_fsl\_sss\_api.h

## Function Documentation

*sss\_status\_t* **sss\_session\_prop\_get\_au8** (*sss\_session\_t* \*session, uint32\_t property, uint8\_t \*pValue, size\_t \*pValueLen)

Get an underlying property of the crypto sub system.

This API is used to get values that are numeric in nature.

Property can be either fixed value that is calculated at compile time and returned directly, or it may involve some access to the underlying system.

**Return**



**Parameters**

- [in] session: Session context
- [in] property: Value that is part of *sss\_session\_prop\_u8\_t*
- [out] pValue: Output buffer array
- [inout] pValueLen: Count of values there are/must be read

**Function *sss\_session\_prop\_get\_u32***

- Defined in file\_sss\_inc\_fsl\_sss\_api.h

**Function Documentation**

*sss\_status\_t* **sss\_session\_prop\_get\_u32** (*sss\_session\_t* \*session, uint32\_t property, uint32\_t \*pValue)

Get an underlying property of the crypto sub system.

This API is used to get values that are numeric in nature.

Property can be either fixed value that is calculated at compile time and returned directly, or it may involve some access to the underlying system.

For applicable properties see *sss\_session\_prop\_u32\_t*

**Return****Parameters**

- [in] session: Session context
- [in] property: Value that is part of *sss\_session\_prop\_u32\_t*
- [out] pValue:

**Function *sss\_status\_sz***

- Defined in file\_sss\_inc\_fsl\_sss\_api.h

**Function Documentation**

const char \***sss\_status\_sz** (*sss\_status\_t* status)

Returns string error code for *sss\_status\_t*.

**Return** String conversion of status to String.

**Parameters**

- [in] status: See *sss\_status\_t*

## Function `sss_symmetric_context_free`

- Defined in `file_sss_inc_fsl_sss_api.h`

### Function Documentation

void **sss\_symmetric\_context\_free** (*sss\_symmetric\_t* \*context)  
Symmetric context release. The function frees symmetric context.

#### Parameters

- context: Pointer to symmetric crypto context.

## Function `sss_symmetric_context_init`

- Defined in `file_sss_inc_fsl_sss_api.h`

### Function Documentation

*sss\_status\_t* **sss\_symmetric\_context\_init** (*sss\_symmetric\_t* \*context, *sss\_session\_t* \*session,  
*sss\_object\_t* \*keyObject, *sss\_algorithm\_t* algorithm,  
*sss\_mode\_t* mode)  
Symmetric context init. The function initializes symmetric context with initial values.

**Return** Status of the operation

#### Parameters

- context: Pointer to symmetric crypto context.
- session: Associate SSS session with symmetric context.
- keyObject: Associate SSS key object with symmetric context.
- algorithm: One of the symmetric algorithms defined by *sss\_algorithm\_t*.
- mode: One of the modes defined by *sss\_mode\_t*.

#### Return Value

- *kStatus\_SSS\_Success*: The operation has completed successfully.
- *kStatus\_SSS\_Fail*: The operation has failed.
- *kStatus\_SSS\_InvalidArgument*: One of the arguments is invalid for the function to execute.

## Function `sss_tunnel`

- Defined in file `_sss_inc_fsl_sss_api.h`

## Function Documentation

*sss\_status\_t* **sss\_tunnel** (*sss\_tunnel\_t* \*context, uint8\_t \*data, size\_t dataLen, *sss\_object\_t* \*keyObjects, uint32\_t keyObjectCount, uint32\_t tunnelType)

Tunnelling service.

### Parameters

- [inout] context: Pointer to tunnel context.
- data: Pointer to data to be send to subsystem.
- dataLen: Length of the data in bytes.
- keyObjects: Objects references used by the service.
- keyObjectCount: Number of key references at keyObjects.
- tunnelType: Implementation specific id of the service.

## Function `sss_tunnel_context_free`

- Defined in file `_sss_inc_fsl_sss_api.h`

## Function Documentation

void **sss\_tunnel\_context\_free** (*sss\_tunnel\_t* \*context)

Destructor for the tunnelling service context.

### Parameters

- [out] context: Pointer to tunnel context.

## Function `sss_tunnel_context_init`

- Defined in file `_sss_inc_fsl_sss_api.h`

## Function Documentation

*sss\_status\_t* **sss\_tunnel\_context\_init** (*sss\_tunnel\_t* \*context, *sss\_session\_t* \*session)

Constructor for the tunnelling service context.

Earlier: `sss_status_t sss_tunnel_context_init( sss_session_t *session, sss_tunnel_t *context);`

Now: Parameters are swapped `sss_status_t sss_tunnel_context_init( sss_tunnel_t *context, sss_session_t *session);`

### Parameters

- [out] context: Pointer to tunnel context. Tunnel context is updated on function return.

- `session`: Pointer to session this tunnelling service belongs to.

## Defines

### Define `ENSURE_OR_BREAK`

- Defined in `file_hostlib_hostLib_inc_nxEnsure.h`

### Define Documentation

#### **`ENSURE_OR_BREAK` (CONDITION)**

If condition fails, break.

Sample Usage:

```
int SomeAPI()
{
 ...

 do {
 status = Operation1();
 ENSURE_OR_BREAK(0 == status);

 status = Operation2();
 ENSURE_OR_BREAK(0 == status);

 ...

 } while(0);

 return status;
}
```

### Define `ENSURE_OR_EXIT_WITH_STATUS_ON_ERROR`

- Defined in `file_hostlib_hostLib_inc_nxEnsure.h`

### Define Documentation

#### **`ENSURE_OR_EXIT_WITH_STATUS_ON_ERROR` (CONDITION, STATUS, RETURN\_VALUE)**

If condition fails, goto quit with return value status updated.

```
int SomeAPI()
{
 int status = 0;
 ...

 value = Operation1();
 ENSURE_OR_QUIT_WITH_STATUS_ON_ERROR(0 == value, status, ERR_FAIL);

 value = Operation2();
 ENSURE_OR_QUIT_WITH_STATUS_ON_ERROR(0 == value, status, ERR_NOT_ENOUGH_SPACE);
}
```

(continues on next page)

(continued from previous page)

```

 ...
quit:
 return status;
}

```

**Warning** This macro introduces system of multiple returns from a function which is not easy to debug/trace through and hence not recommended.

## Define ENSURE\_OR\_GO\_CLEANUP

- Defined in file\_hostlib\_hostLib\_inc\_nxEnsure.h

### Define Documentation

#### ENSURE\_OR\_GO\_CLEANUP (CONDITION)

If condition fails, goto :cleanup label

```

{
 ...

 status = Operation1();
 ENSURE_OR_GO_CLEANUP(0 == status);

 status = Operation2();
 ENSURE_OR_GO_CLEANUP(0 == status);

 ...

cleanup:
 return status;
}

```

## Define ENSURE\_OR\_GO\_EXIT

- Defined in file\_hostlib\_hostLib\_inc\_nxEnsure.h

### Define Documentation

#### ENSURE\_OR\_GO\_EXIT (CONDITION)

If condition fails, goto :exit label

```

{
 ...

 status = Operation1();
 ENSURE_OR_GO_EXIT(0 == status);

 status = Operation2();
 ENSURE_OR_GO_EXIT(0 == status);
}

```

(continues on next page)

(continued from previous page)

```
...

exit:
 return status;
}
```

## Define ENSURE\_OR\_RETURN

- Defined in file\_hostlib\_hostLib\_inc\_nxEnsure.h

## Define Documentation

### ENSURE\_OR\_RETURN (CONDITION)

If condition fails, return

```
void SomeAPI()
{
 ...

 status = Operation1();
 ENSURE_OR_RETURN(0 == status);

 status = Operation2();
 ENSURE_OR_RETURN(0 == status);

 ...

 return;
}
```

**Warning** This macro introduces system of multiple returns from a function which is not easy to debug/trace through and hence not recommended.

## Define ENSURE\_OR\_RETURN\_ON\_ERROR

- Defined in file\_hostlib\_hostLib\_inc\_nxEnsure.h

## Define Documentation

### ENSURE\_OR\_RETURN\_ON\_ERROR (CONDITION, RETURN\_VALUE)

If condition fails, return

```
int SomeAPI()
{
 ...

 status = Operation1();
 ENSURE_OR_RETURN_ON_ERROR(0 == status, ERR_FAIL);
}
```

(continues on next page)

(continued from previous page)

```

 status = Operation2();
 ENSURE_OR_RETURN_ON_ERROR(0 == status, ERR_NOT_ENOUGH_SPACE);

 ...

 return 0;
}

```

**Warning** This macro introduces system of multiple returns from a function which is not easy to debug/trace through and hence not recommended.

## Define EX\_SSS\_BOOT\_OPEN\_HOST\_SESSION

- Defined in file\_sss\_ex\_inc\_ex\_sss\_main\_inc.h

### Define Documentation

**EX\_SSS\_BOOT\_OPEN\_HOST\_SESSION**

## Define EX\_SSS\_BOOT\_RTOS\_STACK\_SIZE

- Defined in file\_sss\_ex\_inc\_ex\_sss\_main\_inc.h

### Define Documentation

**EX\_SSS\_BOOT\_RTOS\_STACK\_SIZE**

## Define kSE05x\_AuthType\_AESKey

- Defined in file\_hostlib\_hostLib\_inc\_nxScp03\_Types.h

### Define Documentation

**kSE05x\_AuthType\_AESKey**

## Define kSE05x\_AuthType\_ECKey

- Defined in file\_hostlib\_hostLib\_inc\_nxScp03\_Types.h

## Define Documentation

**kSE05x\_AuthType\_ECKey**

## Define kSE05x\_AuthType\_None

- Defined in file\_hostlib\_hostLib\_inc\_nxScp03\_Types.h

## Define Documentation

**kSE05x\_AuthType\_None**

## Define kSE05x\_AuthType\_SCP03

- Defined in file\_hostlib\_hostLib\_inc\_nxScp03\_Types.h

## Define Documentation

**kSE05x\_AuthType\_SCP03**

## Define kSE05x\_AuthType\_UserID

- Defined in file\_hostlib\_hostLib\_inc\_nxScp03\_Types.h

## Define Documentation

**kSE05x\_AuthType\_UserID**

## Define kSE05x\_INS\_I2CM\_Attestation

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

## Define Documentation

**kSE05x\_INS\_I2CM\_Attestation**

When we want to read I2CM Data with attestation



### Define kSE05x\_INS\_READ\_With\_Attestation

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

### Define Documentation

#### **kSE05x\_INS\_READ\_With\_Attestation**

When we want to read with attestation

### Define kSSS\_AuthType\_AppletSCP03

- Defined in file\_hostlib\_hostLib\_inc\_nxScp03\_Types.h

### Define Documentation

#### **kSSS\_AuthType\_AppletSCP03**

### Define kSSS\_AuthType\_FastSCP

- Defined in file\_hostlib\_hostLib\_inc\_nxScp03\_Types.h

### Define Documentation

#### **kSSS\_AuthType\_FastSCP**

### Define kSSS\_AuthType\_FastSCP\_Counter

- Defined in file\_hostlib\_hostLib\_inc\_nxScp03\_Types.h

### Define Documentation

#### **kSSS\_AuthType\_FastSCP\_Counter**

### Define kSSS\_AuthType\_INT\_FastSCP\_Counter

- Defined in file\_hostlib\_hostLib\_inc\_nxScp03\_Types.h

## Define Documentation

**kSSS\_AuthType\_INT\_FastSCP\_Counter**

## Define NX\_ENSURE\_DO\_LOG\_MESSAGE

- Defined in file\_hostlib\_hostLib\_inc\_nxEnsure.h

## Define Documentation

**NX\_ENSURE\_DO\_LOG\_MESSAGE**

Build time over-ride if we want to enable/disable Warning Prints

During debug builds, it makes sense to print them, During retail builds, such loggings would be of any use and remove and reduce code size.

## Define NX\_ENSURE\_MESSAGE

- Defined in file\_hostlib\_hostLib\_inc\_nxEnsure.h

## Define Documentation

**NX\_ENSURE\_MESSAGE** (strCONDITION)

Warning print of the parameter strCONDITION

**Warning** NX\_ENSURE\_MESSAGE is an internal message/API to this file. Do not use directly.

## Define PCONTEXT

- Defined in file\_sss\_ex\_inc\_ex\_sss\_main\_inc.h

## Define Documentation

**PCONTEXT**

## Define SE050\_INS\_MASK\_INS\_CHAR

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

## Define Documentation

### **SE050\_INS\_MASK\_INS\_CHAR**

3 MSBit for instruction characteristics.

## Define SE050\_INS\_MASK\_INSTRUCTION

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

## Define Documentation

### **SE050\_INS\_MASK\_INSTRUCTION**

5 LSBit for instruction

## Define SE050\_MAX\_APDU\_PAYLOAD\_LENGTH

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

## Define Documentation

### **SE050\_MAX\_APDU\_PAYLOAD\_LENGTH**

the maximum APDU payload length will be smaller, depending on which protocol applies, etc.

## Define SE050\_MAX\_I2CM\_COMMAND\_LENGTH

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

## Define Documentation

### **SE050\_MAX\_I2CM\_COMMAND\_LENGTH**

How many bytes can be used for buffer for I2C Master interface

## Define SE050\_MAX\_NUMBER\_OF\_SESSIONS

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

## Define Documentation

### **SE050\_MAX\_NUMBER\_OF\_SESSIONS**

Maximum number of session supported by SE050

## Define SE050\_OBJECT\_IDENTIFIER\_SIZE

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

## Define Documentation

### SE050\_OBJECT\_IDENTIFIER\_SIZE

Maximum number of session supported by SE050

## Define Se05x\_API\_ECGenSharedSecret

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU.h

## Define Documentation

### Se05x\_API\_ECGenSharedSecret

Wrapper for *Se05x\_API\_ECDHGenerateSharedSecret*

## Define Se05x\_API\_SHAOneShot

- Defined in file\_hostlib\_hostLib\_se05x\_03\_xx\_xx\_se05x\_APDU.h

## Define Documentation

### Se05x\_API\_SHAOneShot

Wrapper for *Se05x\_API\_DigestOneShot*

## Define se05x\_auth\_context\_t

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_types.h

## Define Documentation

### se05x\_auth\_context\_t

deprecated : Used only for backwards compatibility

## Define SE05x\_AuthCtx\_t

- Defined in file\_hostlib\_hostLib\_inc\_nxScp03\_Types.h

## Define Documentation

**SE05x\_AuthCtx\_t**

## Define SE05x\_AuthType\_t

- Defined in file\_hostlib\_hostLib\_inc\_nxScp03\_Types.h

## Define Documentation

**SE05x\_AuthType\_t**

## Define SE05x\_Connect\_Ctx\_t

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_types.h

## Define Documentation

**SE05x\_Connect\_Ctx\_t**

deprecated : Used only for backwards compatibility

## Define SE05x\_CryptoObjectID\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

## Define Documentation

**SE05x\_CryptoObjectID\_t**

Crypto object identifiers

## Define SE05x\_KeyID\_KEK\_NONE

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

## Define Documentation

**SE05x\_KeyID\_KEK\_NONE**

Case when there is no KEK

### Define SE05x\_KeyID\_MFDF\_NONE

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

### Define Documentation

#### SE05x\_KeyID\_MFDF\_NONE

[Optional: if the authentication key is the same as the key to be replaced, this TAG should not be present].

### Define SE05x\_MaxAttempts\_NA

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

### Define Documentation

#### SE05x\_MaxAttempts\_NA

Identify in code that this is not an AUTH object and hence not applicable

### Define SE05x\_MaxAttempts\_UNLIMITED

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

### Define Documentation

#### SE05x\_MaxAttempts\_UNLIMITED

Fall back to applet default

### Define sss\_aead\_context\_free

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Define Documentation

**sss\_aead\_context\_free** (context)

### Define sss\_aead\_context\_init

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Define Documentation

**sss\_aead\_context\_init** (context, session, keyObject, algorithm, mode)

### Define sss\_aead\_finish

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Define Documentation

**sss\_aead\_finish** (context, srcData, srcLen, destData, destLen, tag, tagLen)

### Define sss\_aead\_init

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Define Documentation

**sss\_aead\_init** (context, nonce, nonceLen, tagLen, aadLen, payloadLen)

### Define sss\_aead\_one\_go

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Define Documentation

**sss\_aead\_one\_go** (context, srcData, destData, size, nonce, nonceLen, aad, aadLen, tag, tagLen)

### Define SSS\_AEAD\_TYPE\_IS\_SE05X

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_types.h

### Define Documentation

**SSS\_AEAD\_TYPE\_IS\_SE05X** (context)

Are we using SE05X as crypto subsystem?

### Define `sss_aead_update`

- Defined in `file_sss_inc_fsl_sss_se05x_apis.h`

### Define Documentation

**`sss_aead_update`** (`context`, `srcData`, `srcLen`, `destData`, `destLen`)

### Define `sss_aead_update_aad`

- Defined in `file_sss_inc_fsl_sss_se05x_apis.h`

### Define Documentation

**`sss_aead_update_aad`** (`context`, `aadData`, `aadDataLen`)

### Define `SSS_AES_BLOCK_SIZE`

- Defined in `file_sss_inc_fsl_sss_api.h`

### Define Documentation

**`SSS_AES_BLOCK_SIZE`**  
Size of an AES Block, in bytes

### Define `SSS_API_VERSION`

- Defined in `file_sss_inc_fsl_sss_api.h`

### Define Documentation

**`SSS_API_VERSION`**  
Version of the SSS API

### Define `sss_asymmetric_context_free`

- Defined in `file_sss_inc_fsl_sss_se05x_apis.h`



### Define Documentation

**sss\_asymmetric\_context\_free** (context)

### Define sss\_asymmetric\_context\_init

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Define Documentation

**sss\_asymmetric\_context\_init** (context, session, keyObject, algorithm, mode)

### Define sss\_asymmetric\_decrypt

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Define Documentation

**sss\_asymmetric\_decrypt** (context, srcData, srcLen, destData, destLen)

### Define sss\_asymmetric\_encrypt

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Define Documentation

**sss\_asymmetric\_encrypt** (context, srcData, srcLen, destData, destLen)

### Define sss\_asymmetric\_sign\_digest

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Define Documentation

**sss\_asymmetric\_sign\_digest** (context, digest, digestLen, signature, signatureLen)

## Define **SSS\_ASYMMETRIC\_TYPE\_IS\_SE05X**

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_types.h

### Define Documentation

**SSS\_ASYMMETRIC\_TYPE\_IS\_SE05X** (context)  
Are we using SE05X as crypto subsystem?

## Define **sss\_asymmetric\_verify\_digest**

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Define Documentation

**sss\_asymmetric\_verify\_digest** (context, digest, digestLen, signature, signatureLen)

## Define **sss\_cipher\_crypt\_ctr**

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Define Documentation

**sss\_cipher\_crypt\_ctr** (context, srcData, destData, size, initialCounter, lastEncryptedCounter, szLeft)

## Define **sss\_cipher\_finish**

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Define Documentation

**sss\_cipher\_finish** (context, srcData, srcLen, destData, destLen)

## Define **sss\_cipher\_init**

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Define Documentation

**sss\_cipher\_init** (context, iv, ivLen)

### Define sss\_cipher\_one\_go

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Define Documentation

**sss\_cipher\_one\_go** (context, iv, ivLen, srcData, destData, dataLen)

### Define sss\_cipher\_update

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Define Documentation

**sss\_cipher\_update** (context, srcData, srcLen, destData, destLen)

### Define sss\_derive\_key\_context\_free

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Define Documentation

**sss\_derive\_key\_context\_free** (context)

### Define sss\_derive\_key\_context\_init

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Define Documentation

**sss\_derive\_key\_context\_init** (context, session, keyObject, algorithm, mode)

### Define `sss_derive_key_dh`

- Defined in `file_sss_inc_fsl_sss_se05x_apis.h`

### Define Documentation

**sss\_derive\_key\_dh** (context, otherPartyKeyObject, derivedKeyObject)

### Define `sss_derive_key_go`

- Defined in `file_sss_inc_fsl_sss_se05x_apis.h`

### Define Documentation

**sss\_derive\_key\_go** (context, saltData, saltLen, info, infoLen, derivedKeyObject, deriveDataLen, hkdfOutput, hkdfOutputLen)

### Define `SSS_DERIVE_KEY_TYPE_IS_SE05X`

- Defined in `file_sss_inc_fsl_sss_se05x_types.h`

### Define Documentation

**SSS\_DERIVE\_KEY\_TYPE\_IS\_SE05X** (context)  
Are we using SE05X as crypto subsystem?

### Define `SSS_DES_BLOCK_SIZE`

- Defined in `file_sss_inc_fsl_sss_api.h`

### Define Documentation

**SSS\_DES\_BLOCK\_SIZE**  
Size of a DES Block, in bytes

### Define `SSS_DES_IV_SIZE`

- Defined in `file_sss_inc_fsl_sss_api.h`

### Define Documentation

#### **SSS\_DES\_IV\_SIZE**

Size of a DES IV, in bytes

### Define SSS\_DES\_KEY\_SIZE

- Defined in file\_sss\_inc\_fsl\_sss\_api.h

### Define Documentation

#### **SSS\_DES\_KEY\_SIZE**

Size of a DES Key, in bytes

### Define sss\_digest\_context\_free

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Define Documentation

**sss\_digest\_context\_free** (context)

### Define sss\_digest\_context\_init

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Define Documentation

**sss\_digest\_context\_init** (context, session, algorithm, mode)

### Define sss\_digest\_finish

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Define Documentation

**sss\_digest\_finish** (context, digest, digestLen)

### Define `sss_digest_init`

- Defined in `file_sss_inc_fsl_sss_se05x_apis.h`

### Define Documentation

**`sss_digest_init`** (context)

### Define `sss_digest_one_go`

- Defined in `file_sss_inc_fsl_sss_se05x_apis.h`

### Define Documentation

**`sss_digest_one_go`** (context, message, messageLen, digest, digestLen)

### Define `SSS_DIGEST_TYPE_IS_SE05X`

- Defined in `file_sss_inc_fsl_sss_se05x_types.h`

### Define Documentation

**`SSS_DIGEST_TYPE_IS_SE05X`** (context)  
Are we using SE05X as crypto subsystem?

### Define `sss_digest_update`

- Defined in `file_sss_inc_fsl_sss_se05x_apis.h`

### Define Documentation

**`sss_digest_update`** (context, message, messageLen)

### Define `SSS_ENUM`

- Defined in `file_sss_inc_fsl_sss_api.h`

## Define Documentation

**SSS\_ENUM** (GROUP, INDEX)  
Helper macro to set enum value

## Define **sss\_key\_object\_allocate\_handle**

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

## Define Documentation

**sss\_key\_object\_allocate\_handle** (keyObject, keyId, keyPart, cipherType, keyByteLenMax, options)

## Define **sss\_key\_object\_free**

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

## Define Documentation

**sss\_key\_object\_free** (keyObject)

## Define **sss\_key\_object\_get\_access**

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

## Define Documentation

**sss\_key\_object\_get\_access** (keyObject, access)

## Define **sss\_key\_object\_get\_handle**

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

## Define Documentation

**sss\_key\_object\_get\_handle** (keyObject, keyId)

### Define `sss_key_object_get_purpose`

- Defined in `file_sss_inc_fsl_sss_se05x_apis.h`

### Define Documentation

**`sss_key_object_get_purpose`** (`keyObject`, `purpose`)

### Define `sss_key_object_get_user`

- Defined in `file_sss_inc_fsl_sss_se05x_apis.h`

### Define Documentation

**`sss_key_object_get_user`** (`keyObject`, `user`)

### Define `sss_key_object_init`

- Defined in `file_sss_inc_fsl_sss_se05x_apis.h`

### Define Documentation

**`sss_key_object_init`** (`keyObject`, `keyStore`)

### Define `sss_key_object_set_access`

- Defined in `file_sss_inc_fsl_sss_se05x_apis.h`

### Define Documentation

**`sss_key_object_set_access`** (`keyObject`, `access`, `options`)

### Define `sss_key_object_set_eccgfp_group`

- Defined in `file_sss_inc_fsl_sss_se05x_apis.h`



### Define Documentation

**sss\_key\_object\_set\_eccgfp\_group** (keyObject, group)

### Define sss\_key\_object\_set\_purpose

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Define Documentation

**sss\_key\_object\_set\_purpose** (keyObject, purpose, options)

### Define sss\_key\_object\_set\_user

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Define Documentation

**sss\_key\_object\_set\_user** (keyObject, user, options)

### Define sss\_key\_store\_allocate

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Define Documentation

**sss\_key\_store\_allocate** (keyStore, keyStoreId)

### Define sss\_key\_store\_context\_free

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Define Documentation

**sss\_key\_store\_context\_free** (keyStore)

### Define `sss_key_store_context_init`

- Defined in `file_sss_inc_fsl_sss_se05x_apis.h`

### Define Documentation

**`sss_key_store_context_init`** (`keyStore`, `session`)

### Define `sss_key_store_erase_key`

- Defined in `file_sss_inc_fsl_sss_se05x_apis.h`

### Define Documentation

**`sss_key_store_erase_key`** (`keyStore`, `keyObject`)

### Define `sss_key_store_freeze_key`

- Defined in `file_sss_inc_fsl_sss_se05x_apis.h`

### Define Documentation

**`sss_key_store_freeze_key`** (`keyStore`, `keyObject`)

### Define `sss_key_store_generate_key`

- Defined in `file_sss_inc_fsl_sss_se05x_apis.h`

### Define Documentation

**`sss_key_store_generate_key`** (`keyStore`, `keyObject`, `keyBitLen`, `options`)

### Define `sss_key_store_get_key`

- Defined in `file_sss_inc_fsl_sss_se05x_apis.h`

### Define Documentation

**sss\_key\_store\_get\_key** (keyStore, keyObject, data, dataLen, pKeyBitLen)

### Define sss\_key\_store\_load

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Define Documentation

**sss\_key\_store\_load** (keyStore)

### Define sss\_key\_store\_open\_key

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Define Documentation

**sss\_key\_store\_open\_key** (keyStore, keyObject)

### Define sss\_key\_store\_save

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Define Documentation

**sss\_key\_store\_save** (keyStore)

### Define sss\_key\_store\_set\_key

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Define Documentation

**sss\_key\_store\_set\_key** (keyStore, keyObject, data, dataLen, keyBitLen, options, optionsLen)

## Define `SSS_KEY_STORE_TYPE_IS_SE05X`

- Defined in `file_sss_inc_fsl_sss_se05x_types.h`

## Define Documentation

**SSS\_KEY\_STORE\_TYPE\_IS\_SE05X** (keyStore)  
Are we using SE05X as crypto subsystem?

## Define `sss_mac_context_free`

- Defined in `file_sss_inc_fsl_sss_se05x_apis.h`

## Define Documentation

**sss\_mac\_context\_free** (context)

## Define `sss_mac_context_init`

- Defined in `file_sss_inc_fsl_sss_se05x_apis.h`

## Define Documentation

**sss\_mac\_context\_init** (context, session, keyObject, algorithm, mode)

## Define `sss_mac_finish`

- Defined in `file_sss_inc_fsl_sss_se05x_apis.h`

## Define Documentation

**sss\_mac\_finish** (context, mac, macLen)

## Define `sss_mac_init`

- Defined in `file_sss_inc_fsl_sss_se05x_apis.h`

### Define Documentation

**sss\_mac\_init** (context)

### Define sss\_mac\_one\_go

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Define Documentation

**sss\_mac\_one\_go** (context, message, messageLen, mac, macLen)

### Define SSS\_MAC\_TYPE\_IS\_SE05X

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_types.h

### Define Documentation

**SSS\_MAC\_TYPE\_IS\_SE05X** (context)  
Are we using SE05X as crypto subsystem?

### Define sss\_mac\_update

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Define Documentation

**sss\_mac\_update** (context, message, messageLen)

### Define SSS\_OBJECT\_TYPE\_IS\_SE05X

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_types.h

### Define Documentation

**SSS\_OBJECT\_TYPE\_IS\_SE05X** (pObject)  
Are we using SE05X as crypto subsystem?

### Define `sss_rng_context_free`

- Defined in `file_sss_inc_fsl_sss_se05x_apis.h`

### Define Documentation

**`sss_rng_context_free`** (context)

### Define `sss_rng_context_init`

- Defined in `file_sss_inc_fsl_sss_se05x_apis.h`

### Define Documentation

**`sss_rng_context_init`** (context, session)

### Define `SSS_RNG_CONTEXT_TYPE_IS_SE05X`

- Defined in `file_sss_inc_fsl_sss_se05x_types.h`

### Define Documentation

**`SSS_RNG_CONTEXT_TYPE_IS_SE05X`** (context)  
Are we using SE05X as crypto subsystem?

### Define `sss_rng_get_random`

- Defined in `file_sss_inc_fsl_sss_se05x_apis.h`

### Define Documentation

**`sss_rng_get_random`** (context, random\_data, dataLen)

### Define `sss_session_close`

- Defined in `file_sss_inc_fsl_sss_se05x_apis.h`

### Define Documentation

**sss\_session\_close** (session)

### Define sss\_session\_create

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Define Documentation

**sss\_session\_create** (session, subsystem, application\_id, connection\_type, connectionData)

### Define sss\_session\_delete

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Define Documentation

**sss\_session\_delete** (session)

### Define sss\_session\_open

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Define Documentation

**sss\_session\_open** (session, subsystem, application\_id, connection\_type, connectionData)

### Define sss\_session\_prop\_get\_au8

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Define Documentation

**sss\_session\_prop\_get\_au8** (session, property, pValue, pValueLen)

### Define `sss_session_prop_get_u32`

- Defined in `file_sss_inc_fsl_sss_se05x_apis.h`

### Define Documentation

**`sss_session_prop_get_u32`** (session, property, pValue)

### Define `SSS_SESSION_TYPE_IS_SE05X`

- Defined in `file_sss_inc_fsl_sss_se05x_types.h`

### Define Documentation

**`SSS_SESSION_TYPE_IS_SE05X`** (session)  
Are we using SE05X as crypto subsystem?

### Define `SSS_SUBSYSTEM_TYPE_IS_SE05X`

- Defined in `file_sss_inc_fsl_sss_se05x_types.h`

### Define Documentation

**`SSS_SUBSYSTEM_TYPE_IS_SE05X`** (subsystem)  
Are we using SE05X as crypto subsystem?

### Define `sss_symmetric_context_free`

- Defined in `file_sss_inc_fsl_sss_se05x_apis.h`

### Define Documentation

**`sss_symmetric_context_free`** (context)

### Define `sss_symmetric_context_init`

- Defined in `file_sss_inc_fsl_sss_se05x_apis.h`



### Define Documentation

**sss\_symmetric\_context\_init** (context, session, keyObject, algorithm, mode)

### Define SSS\_SYMMETRIC\_TYPE\_IS\_SE05X

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_types.h

### Define Documentation

**SSS\_SYMMETRIC\_TYPE\_IS\_SE05X** (context)  
Are we using SE05X as crypto subsystem?

### Define sss\_tunnel

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Define Documentation

**sss\_tunnel** (context, data, dataLen, keyObjects, keyObjectCount, tunnelType)

### Define sss\_tunnel\_context\_free

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Define Documentation

**sss\_tunnel\_context\_free** (context)

### Define sss\_tunnel\_context\_init

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_apis.h

### Define Documentation

**sss\_tunnel\_context\_init** (context, session)

## Define SSS\_TUNNEL\_CONTEXT\_TYPE\_IS\_SE05X

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_types.h

### Define Documentation

**SSS\_TUNNEL\_CONTEXT\_TYPE\_IS\_SE05X** (context)  
Are we using SE05X as crypto subsystem?

## Define SSS\_TUNNEL\_TYPE\_IS\_SE05X

- Defined in file\_sss\_inc\_fsl\_sss\_se05x\_types.h

### Define Documentation

**SSS\_TUNNEL\_TYPE\_IS\_SE05X** (context)  
Are we using SE05X as crypto subsystem?

## Typedefs

### Typedef SE05x\_KeyID\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

### Typedef Documentation

**typedef** uint32\_t **SE05x\_KeyID\_t**  
SE05X's key IDs

### Typedef SE05x\_MacOperation\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

### Typedef Documentation

**typedef** *SE05x\_MACAlgo\_t* **SE05x\_MacOperation\_t**  
HMAC/CMAC Algorithms

### Typedef SE05x\_MaxAttempts\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

### Typedef Documentation

**typedef** uint16\_t **SE05x\_MaxAttempts\_t**  
SE05X key's max attempts

### Typedef SE05x\_SecureObjectType\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

### Typedef Documentation

**typedef** *SE05x\_SecObjTyp\_t* **SE05x\_SecureObjectType\_t**  
Type of Object

### Typedef SE05x\_Variant\_t

- Defined in file\_hostlib\_hostLib\_inc\_se05x\_enums.h

### Typedef Documentation

**typedef** *SE05x\_AppletConfig\_t* **SE05x\_Variant\_t**  
Features which are available / enabled in the Applet



## INDICES AND TABLES

- `genindex`
- `search`



## A

A71CH\_AUTH  
 command line option, 113  
 Applet  
 command line option, 110

## C

CMAKE\_BUILD\_TYPE  
 command line option, 113  
 command line option  
 A71CH\_AUTH, 113  
 Applet, 110  
 CMAKE\_BUILD\_TYPE, 113  
 Host, 110  
 HostCrypto, 111  
 Log, 113  
 mbedTLS\_ALT, 112  
 NXPInternal, 116  
 RTOS, 112  
 SCP, 112  
 SE05X\_Auth, 112  
 SE05X\_Ver, 110  
 SIMW\_INSTALL\_INC\_DIR, 116  
 SIMW\_INSTALL\_SHARE\_DIR, 116  
 SMCOM, 111  
 SSSFTR\_SE05X\_AES, 113  
 SSSFTR\_SE05X\_AuthECKey, 114  
 SSSFTR\_SE05X\_AuthSession, 114  
 SSSFTR\_SE05X\_CREATE\_DELETE\_CRYPTOOBJ,  
 114  
 SSSFTR\_SE05X\_ECC, 113  
 SSSFTR\_SE05X\_KEY\_GET, 114  
 SSSFTR\_SE05X\_KEY\_SET, 114  
 SSSFTR\_SE05X\_RSA, 113  
 SSSFTR\_SW\_AES, 114  
 SSSFTR\_SW\_ECC, 114  
 SSSFTR\_SW\_KEY\_GET, 114  
 SSSFTR\_SW\_KEY\_SET, 114  
 SSSFTR\_SW\_RSA, 114  
 SSSFTR\_SW\_TESTCOUNTERPART, 114  
 WithCodeCoverage, 116  
 WithExtCustomerCode, 116

WithNXPNFCRdLib, 116  
 WithOPCUA\_open62541, 116  
 WithSharedLIB, 116

## E

ENSURE\_OR\_BREAK (*C macro*), 632  
 ENSURE\_OR\_EXIT\_WITH\_STATUS\_ON\_ERROR (*C macro*), 632  
 ENSURE\_OR\_GO\_CLEANUP (*C macro*), 633  
 ENSURE\_OR\_GO\_EXIT (*C macro*), 633  
 ENSURE\_OR\_RETURN (*C macro*), 634  
 ENSURE\_OR\_RETURN\_ON\_ERROR (*C macro*), 634  
 EX\_SSS\_BOOT\_OPEN\_HOST\_SESSION (*C macro*),  
 635  
 EX\_SSS\_BOOT\_RTOS\_STACK\_SIZE (*C macro*), 635

## H

Host  
 command line option, 110  
 HostCrypto  
 command line option, 111

## K

kAccessPermission\_SSS\_ChangeAttributes  
 (*C++ enumerator*), 461  
 kAccessPermission\_SSS\_Delete (*C++ enumerator*), 461  
 kAccessPermission\_SSS\_Read (*C++ enumerator*), 460  
 kAccessPermission\_SSS\_Use (*C++ enumerator*),  
 460  
 kAccessPermission\_SSS\_Write (*C++ enumerator*), 460  
 kAlgorithm\_None (*C++ enumerator*), 461  
 kAlgorithm\_SSS\_AES\_CBC (*C++ enumerator*), 461  
 kAlgorithm\_SSS\_AES\_CCM (*C++ enumerator*), 461  
 kAlgorithm\_SSS\_AES\_CTR (*C++ enumerator*), 461  
 kAlgorithm\_SSS\_AES\_ECB (*C++ enumerator*), 461  
 kAlgorithm\_SSS\_AES\_GCM (*C++ enumerator*), 461  
 kAlgorithm\_SSS\_CHACHA\_POLY (*C++ enumerator*), 461

[kAlgorithm\\_SSS\\_CMACE \(C++ enumerator\), 461](#)  
[kAlgorithm\\_SSS\\_DES3\\_CBC \(C++ enumerator\), 461](#)  
[kAlgorithm\\_SSS\\_DES3\\_ECB \(C++ enumerator\), 461](#)  
[kAlgorithm\\_SSS\\_DES\\_CBC \(C++ enumerator\), 461](#)  
[kAlgorithm\\_SSS\\_DES\\_ECB \(C++ enumerator\), 461](#)  
[kAlgorithm\\_SSS\\_DH \(C++ enumerator\), 461](#)  
[kAlgorithm\\_SSS\\_DSA\\_SHA1 \(C++ enumerator\), 462](#)  
[kAlgorithm\\_SSS\\_DSA\\_SHA224 \(C++ enumerator\), 462](#)  
[kAlgorithm\\_SSS\\_DSA\\_SHA256 \(C++ enumerator\), 462](#)  
[kAlgorithm\\_SSS\\_ECDH \(C++ enumerator\), 461](#)  
[kAlgorithm\\_SSS\\_ECDSA\\_SHA1 \(C++ enumerator\), 462](#)  
[kAlgorithm\\_SSS\\_ECDSA\\_SHA224 \(C++ enumerator\), 462](#)  
[kAlgorithm\\_SSS\\_ECDSA\\_SHA256 \(C++ enumerator\), 462](#)  
[kAlgorithm\\_SSS\\_ECDSA\\_SHA384 \(C++ enumerator\), 462](#)  
[kAlgorithm\\_SSS\\_ECDSA\\_SHA512 \(C++ enumerator\), 462](#)  
[kAlgorithm\\_SSS\\_HMAC\\_SHA1 \(C++ enumerator\), 461](#)  
[kAlgorithm\\_SSS\\_HMAC\\_SHA224 \(C++ enumerator\), 461](#)  
[kAlgorithm\\_SSS\\_HMAC\\_SHA256 \(C++ enumerator\), 461](#)  
[kAlgorithm\\_SSS\\_HMAC\\_SHA384 \(C++ enumerator\), 461](#)  
[kAlgorithm\\_SSS\\_HMAC\\_SHA512 \(C++ enumerator\), 461](#)  
[kAlgorithm\\_SSS\\_RSAES\\_PKCS1\\_OAEP\\_SHA1 \(C++ enumerator\), 462](#)  
[kAlgorithm\\_SSS\\_RSAES\\_PKCS1\\_OAEP\\_SHA224 \(C++ enumerator\), 462](#)  
[kAlgorithm\\_SSS\\_RSAES\\_PKCS1\\_OAEP\\_SHA256 \(C++ enumerator\), 462](#)  
[kAlgorithm\\_SSS\\_RSAES\\_PKCS1\\_OAEP\\_SHA384 \(C++ enumerator\), 462](#)  
[kAlgorithm\\_SSS\\_RSAES\\_PKCS1\\_OAEP\\_SHA512 \(C++ enumerator\), 462](#)  
[kAlgorithm\\_SSS\\_RSAES\\_PKCS1\\_V1\\_5\\_SHA1 \(C++ enumerator\), 462](#)  
[kAlgorithm\\_SSS\\_RSAES\\_PKCS1\\_V1\\_5\\_SHA224 \(C++ enumerator\), 462](#)  
[kAlgorithm\\_SSS\\_RSAES\\_PKCS1\\_V1\\_5\\_SHA256 \(C++ enumerator\), 462](#)  
[kAlgorithm\\_SSS\\_RSAES\\_PKCS1\\_V1\\_5\\_SHA384 \(C++ enumerator\), 462](#)  
[kAlgorithm\\_SSS\\_RSAES\\_PKCS1\\_V1\\_5\\_SHA512 \(C++ enumerator\), 462](#)  
[kAlgorithm\\_SSS\\_RSASSA\\_NO\\_PADDING \(C++ enumerator\), 462](#)  
[kAlgorithm\\_SSS\\_RSASSA\\_PKCS1\\_PSS\\_MGF1\\_SHA1 \(C++ enumerator\), 462](#)  
[kAlgorithm\\_SSS\\_RSASSA\\_PKCS1\\_PSS\\_MGF1\\_SHA224 \(C++ enumerator\), 462](#)  
[kAlgorithm\\_SSS\\_RSASSA\\_PKCS1\\_PSS\\_MGF1\\_SHA256 \(C++ enumerator\), 462](#)  
[kAlgorithm\\_SSS\\_RSASSA\\_PKCS1\\_PSS\\_MGF1\\_SHA384 \(C++ enumerator\), 462](#)  
[kAlgorithm\\_SSS\\_RSASSA\\_PKCS1\\_PSS\\_MGF1\\_SHA512 \(C++ enumerator\), 462](#)  
[kAlgorithm\\_SSS\\_RSASSA\\_PKCS1\\_V1\\_5\\_NO\\_HASH \(C++ enumerator\), 462](#)  
[kAlgorithm\\_SSS\\_RSASSA\\_PKCS1\\_V1\\_5\\_SHA1 \(C++ enumerator\), 462](#)  
[kAlgorithm\\_SSS\\_RSASSA\\_PKCS1\\_V1\\_5\\_SHA224 \(C++ enumerator\), 462](#)  
[kAlgorithm\\_SSS\\_RSASSA\\_PKCS1\\_V1\\_5\\_SHA256 \(C++ enumerator\), 462](#)  
[kAlgorithm\\_SSS\\_RSASSA\\_PKCS1\\_V1\\_5\\_SHA384 \(C++ enumerator\), 462](#)  
[kAlgorithm\\_SSS\\_RSASSA\\_PKCS1\\_V1\\_5\\_SHA512 \(C++ enumerator\), 462](#)  
[kAlgorithm\\_SSS\\_SHA1 \(C++ enumerator\), 461](#)  
[kAlgorithm\\_SSS\\_SHA224 \(C++ enumerator\), 461](#)  
[kAlgorithm\\_SSS\\_SHA256 \(C++ enumerator\), 461](#)  
[kAlgorithm\\_SSS\\_SHA384 \(C++ enumerator\), 461](#)  
[kAlgorithm\\_SSS\\_SHA512 \(C++ enumerator\), 461](#)  
[kKeyObject\\_Mode\\_None \(C++ enumerator\), 464](#)  
[kKeyObject\\_Mode\\_Persistent \(C++ enumerator\), 464](#)  
[kKeyObject\\_Mode\\_Transient \(C++ enumerator\), 464](#)  
[kMode\\_SSS\\_ComputeSharedSecret \(C++ enumerator\), 465](#)  
[kMode\\_SSS\\_Decrypt \(C++ enumerator\), 465](#)  
[kMode\\_SSS\\_Digest \(C++ enumerator\), 465](#)  
[kMode\\_SSS\\_Encrypt \(C++ enumerator\), 465](#)  
[kMode\\_SSS\\_HKDF\\_ExpandOnly \(C++ enumerator\), 465](#)  
[kMode\\_SSS\\_Mac \(C++ enumerator\), 465](#)  
[kMode\\_SSS\\_Sign \(C++ enumerator\), 465](#)  
[kMode\\_SSS\\_Verify \(C++ enumerator\), 465](#)  
[KPolicy\\_Asym\\_Key \(C++ enumerator\), 466](#)  
[KPolicy\\_Common \(C++ enumerator\), 466](#)  
[KPolicy\\_Common\\_PCR\\_Value \(C++ enumerator\), 466](#)  
[KPolicy\\_Counter \(C++ enumerator\), 466](#)  
[KPolicy\\_File \(C++ enumerator\), 466](#)  
[KPolicy\\_None \(C++ enumerator\), 466](#)  
[KPolicy\\_PCR \(C++ enumerator\), 466](#)



KPolicy\_Session (C++ *enumerator*), 466  
 KPolicy\_Sym\_Key (C++ *enumerator*), 466  
 KPolicy\_UserID (C++ *enumerator*), 466  
 kSE05x\_AeadAlgo (C++ *enumerator*), 430  
 kSE05x\_AeadAlgo\_NA (C++ *enumerator*), 430  
 kSE05x\_AppletConfig\_AES (C++ *enumerator*), 431  
 kSE05x\_AppletConfig\_DES (C++ *enumerator*), 431  
 kSE05x\_AppletConfig\_DH\_MONT (C++ *enumerator*), 431  
 kSE05x\_AppletConfig\_ECDSA (C++ *enumerator*), 430  
 kSE05x\_AppletConfig\_ECDSA\_ECDH\_ECDHE (C++ *enumerator*), 430  
 kSE05x\_AppletConfig\_EDDSA (C++ *enumerator*), 430  
 kSE05x\_AppletConfig\_HMAC (C++ *enumerator*), 431  
 kSE05x\_AppletConfig\_I2CM (C++ *enumerator*), 431  
 kSE05x\_AppletConfig\_MIFARE (C++ *enumerator*), 431  
 kSE05x\_AppletConfig\_NA (C++ *enumerator*), 430  
 kSE05x\_AppletConfig\_PBKDF (C++ *enumerator*), 431  
 kSE05x\_AppletConfig\_RFU1 (C++ *enumerator*), 431  
 kSE05x\_AppletConfig\_RFU2 (C++ *enumerator*), 431  
 kSE05x\_AppletConfig\_RSA\_CRT (C++ *enumerator*), 431  
 kSE05x\_AppletConfig\_RSA\_PLAIN (C++ *enumerator*), 431  
 kSE05x\_AppletConfig\_TLS (C++ *enumerator*), 431  
 kSE05x\_AppletResID\_FACTORY\_RESET (C++ *enumerator*), 432  
 kSE05x\_AppletResID\_FEATURE (C++ *enumerator*), 432  
 kSE05x\_AppletResID\_I2CM\_ACCESS (C++ *enumerator*), 432  
 kSE05x\_AppletResID\_KP\_ECKEY\_IMPORT (C++ *enumerator*), 431  
 kSE05x\_AppletResID\_KP\_ECKEY\_USER (C++ *enumerator*), 431  
 kSE05x\_AppletResID\_NA (C++ *enumerator*), 431  
 kSE05x\_AppletResID\_PLATFORM\_SCP (C++ *enumerator*), 432  
 kSE05x\_AppletResID\_RESTRICT (C++ *enumerator*), 432  
 kSE05x\_AppletResID\_TRANSPORT (C++ *enumerator*), 431  
 kSE05x\_AppletResID\_UNIQUE\_ID (C++ *enumerator*), 432  
 kSE05x\_AttestationAlgo\_EC\_PLAIN (C++ *enumerator*), 432  
 kSE05x\_AttestationAlgo\_EC\_SHA (C++ *enumerator*), 432  
 kSE05x\_AttestationAlgo\_EC\_SHA\_224 (C++ *enumerator*), 432  
 kSE05x\_AttestationAlgo\_EC\_SHA\_256 (C++ *enumerator*), 432  
 kSE05x\_AttestationAlgo\_EC\_SHA\_384 (C++ *enumerator*), 432  
 kSE05x\_AttestationAlgo\_EC\_SHA\_512 (C++ *enumerator*), 432  
 kSE05x\_AttestationAlgo\_ECDSA (C++ *enumerator*), 432  
 kSE05x\_AttestationAlgo\_ED25519PH\_SHA\_512 (C++ *enumerator*), 432  
 kSE05x\_AttestationAlgo\_NA (C++ *enumerator*), 432  
 kSE05x\_AttestationAlgo\_RSA\_SHA1\_PKCS1\_PSS (C++ *enumerator*), 432  
 kSE05x\_AttestationAlgo\_RSA\_SHA224\_PKCS1\_PSS (C++ *enumerator*), 432  
 kSE05x\_AttestationAlgo\_RSA\_SHA256\_PKCS1\_PSS (C++ *enumerator*), 432  
 kSE05x\_AttestationAlgo\_RSA\_SHA384\_PKCS1\_PSS (C++ *enumerator*), 432  
 kSE05x\_AttestationAlgo\_RSA\_SHA512\_PKCS1\_PSS (C++ *enumerator*), 433  
 kSE05x\_AttestationAlgo\_RSA\_SHA\_224\_PKCS1 (C++ *enumerator*), 433  
 kSE05x\_AttestationAlgo\_RSA\_SHA\_256\_PKCS1 (C++ *enumerator*), 433  
 kSE05x\_AttestationAlgo\_RSA\_SHA\_384\_PKCS1 (C++ *enumerator*), 433  
 kSE05x\_AttestationAlgo\_RSA\_SHA\_512\_PKCS1 (C++ *enumerator*), 433  
 kSE05x\_AttestationType\_AUTH (C++ *enumerator*), 433  
 kSE05x\_AttestationType\_None (C++ *enumerator*), 433  
 kSE05x\_AuthType\_AESKey (C *macro*), 635  
 kSE05x\_AuthType\_ECKEY (C *macro*), 636  
 kSE05x\_AuthType\_None (C *macro*), 636  
 kSE05x\_AuthType\_SCP03 (C *macro*), 636  
 kSE05x\_AuthType\_UserID (C *macro*), 636  
 kSE05x\_Cipher\_Oper\_Decrypt (C++ *enumerator*), 434  
 kSE05x\_Cipher\_Oper\_Encrypt (C++ *enumerator*), 434  
 kSE05x\_Cipher\_Oper\_NA (C++ *enumerator*), 434  
 kSE05x\_Cipher\_Oper\_OneShot\_Decrypt (C++ *enumerator*), 433  
 kSE05x\_Cipher\_Oper\_OneShot\_Encrypt (C++

*enumerator*), 433  
 kSE05x\_Cipher\_Oper\_OneShot\_NA (C++ *enumerator*), 433  
 kSE05x\_CipherMode\_AES\_CBC\_ISO9797\_M1 (C++ *enumerator*), 434  
 kSE05x\_CipherMode\_AES\_CBC\_ISO9797\_M2 (C++ *enumerator*), 434  
 kSE05x\_CipherMode\_AES\_CBC\_NOPAD (C++ *enumerator*), 434  
 kSE05x\_CipherMode\_AES\_CBC\_PKCS5 (C++ *enumerator*), 435  
 kSE05x\_CipherMode\_AES\_CTR (C++ *enumerator*), 435  
 kSE05x\_CipherMode\_AES\_ECB\_NOPAD (C++ *enumerator*), 434  
 kSE05x\_CipherMode\_AES\_GCM (C++ *enumerator*), 435  
 kSE05x\_CipherMode\_DES\_CBC\_ISO9797\_M1 (C++ *enumerator*), 434  
 kSE05x\_CipherMode\_DES\_CBC\_ISO9797\_M2 (C++ *enumerator*), 434  
 kSE05x\_CipherMode\_DES\_CBC\_NOPAD (C++ *enumerator*), 434  
 kSE05x\_CipherMode\_DES\_CBC\_PKCS5 (C++ *enumerator*), 434  
 kSE05x\_CipherMode\_DES\_ECB\_ISO9797\_M1 (C++ *enumerator*), 434  
 kSE05x\_CipherMode\_DES\_ECB\_ISO9797\_M2 (C++ *enumerator*), 434  
 kSE05x\_CipherMode\_DES\_ECB\_NOPAD (C++ *enumerator*), 434  
 kSE05x\_CipherMode\_DES\_ECB\_PKCS5 (C++ *enumerator*), 434  
 kSE05x\_CipherMode\_NA (C++ *enumerator*), 434  
 kSE05x\_CryptoContext\_AEAD (C++ *enumerator*), 435  
 kSE05x\_CryptoContext\_CIPHER (C++ *enumerator*), 435  
 kSE05x\_CryptoContext\_DIGEST (C++ *enumerator*), 435  
 kSE05x\_CryptoContext\_NA (C++ *enumerator*), 435  
 kSE05x\_CryptoContext\_SIGNATURE (C++ *enumerator*), 435  
 kSE05x\_CryptoObject\_AES\_CBC\_ISO9797\_M1 (C++ *enumerator*), 436  
 kSE05x\_CryptoObject\_AES\_CBC\_ISO9797\_M2 (C++ *enumerator*), 436  
 kSE05x\_CryptoObject\_AES\_CBC\_NOPAD (C++ *enumerator*), 436  
 kSE05x\_CryptoObject\_AES\_CBC\_PKCS5 (C++ *enumerator*), 436  
 kSE05x\_CryptoObject\_AES\_CTR (C++ *enumerator*), 436  
 kSE05x\_CryptoObject\_AES\_ECB\_NOPAD (C++ *enumerator*), 436  
 kSE05x\_CryptoObject\_AES\_GCM (C++ *enumerator*), 436  
 kSE05x\_CryptoObject\_CMAC\_128 (C++ *enumerator*), 436  
 kSE05x\_CryptoObject\_DES\_CBC\_ISO9797\_M1 (C++ *enumerator*), 436  
 kSE05x\_CryptoObject\_DES\_CBC\_ISO9797\_M2 (C++ *enumerator*), 436  
 kSE05x\_CryptoObject\_DES\_CBC\_NOPAD (C++ *enumerator*), 436  
 kSE05x\_CryptoObject\_DES\_CBC\_PKCS5 (C++ *enumerator*), 436  
 kSE05x\_CryptoObject\_DES\_ECB\_ISO9797\_M1 (C++ *enumerator*), 436  
 kSE05x\_CryptoObject\_DES\_ECB\_ISO9797\_M2 (C++ *enumerator*), 436  
 kSE05x\_CryptoObject\_DES\_ECB\_NOPAD (C++ *enumerator*), 436  
 kSE05x\_CryptoObject\_DES\_ECB\_PKCS5 (C++ *enumerator*), 436  
 kSE05x\_CryptoObject\_DIGEST\_SHA (C++ *enumerator*), 435  
 kSE05x\_CryptoObject\_DIGEST\_SHA224 (C++ *enumerator*), 435  
 kSE05x\_CryptoObject\_DIGEST\_SHA256 (C++ *enumerator*), 435  
 kSE05x\_CryptoObject\_DIGEST\_SHA384 (C++ *enumerator*), 435  
 kSE05x\_CryptoObject\_DIGEST\_SHA512 (C++ *enumerator*), 436  
 kSE05x\_CryptoObject\_HMAC\_SHA1 (C++ *enumerator*), 436  
 kSE05x\_CryptoObject\_HMAC\_SHA256 (C++ *enumerator*), 436  
 kSE05x\_CryptoObject\_HMAC\_SHA384 (C++ *enumerator*), 436  
 kSE05x\_CryptoObject\_HMAC\_SHA512 (C++ *enumerator*), 436  
 kSE05x\_CryptoObject\_NA (C++ *enumerator*), 435  
 kSE05x\_DigestMode\_NA (C++ *enumerator*), 436  
 kSE05x\_DigestMode\_NO\_HASH (C++ *enumerator*), 436  
 kSE05x\_DigestMode\_SHA (C++ *enumerator*), 436  
 kSE05x\_DigestMode\_SHA224 (C++ *enumerator*), 436  
 kSE05x\_DigestMode\_SHA256 (C++ *enumerator*), 436  
 kSE05x\_DigestMode\_SHA384 (C++ *enumerator*), 437  
 kSE05x\_DigestMode\_SHA512 (C++ *enumerator*), 437  
 kSE05x\_ECCurve\_Brainpool160 (C++ *enumerator*), 437

tor), 437

kSE05x\_ECCurve\_Brainpool192 (C++ enumerator), 437

kSE05x\_ECCurve\_Brainpool224 (C++ enumerator), 437

kSE05x\_ECCurve\_Brainpool256 (C++ enumerator), 437

kSE05x\_ECCurve\_Brainpool320 (C++ enumerator), 437

kSE05x\_ECCurve\_Brainpool384 (C++ enumerator), 437

kSE05x\_ECCurve\_Brainpool512 (C++ enumerator), 437

kSE05x\_ECCurve\_ECC\_ED\_25519 (C++ enumerator), 437

kSE05x\_ECCurve\_ECC\_MONT\_DH\_25519 (C++ enumerator), 437

kSE05x\_ECCurve\_ECC\_MONT\_DH\_448 (C++ enumerator), 437

kSE05x\_ECCurve\_NA (C++ enumerator), 437

kSE05x\_ECCurve\_NIST\_P192 (C++ enumerator), 437

kSE05x\_ECCurve\_NIST\_P224 (C++ enumerator), 437

kSE05x\_ECCurve\_NIST\_P256 (C++ enumerator), 437

kSE05x\_ECCurve\_NIST\_P384 (C++ enumerator), 437

kSE05x\_ECCurve\_NIST\_P521 (C++ enumerator), 437

kSE05x\_ECCurve\_Secp160k1 (C++ enumerator), 437

kSE05x\_ECCurve\_Secp192k1 (C++ enumerator), 437

kSE05x\_ECCurve\_Secp224k1 (C++ enumerator), 437

kSE05x\_ECCurve\_Secp256k1 (C++ enumerator), 437

kSE05x\_ECCurve\_TPM\_ECC\_BN\_P256 (C++ enumerator), 437

kSE05x\_ECCurveParam\_NA (C++ enumerator), 438

kSE05x\_ECCurveParam\_PARAM\_A (C++ enumerator), 438

kSE05x\_ECCurveParam\_PARAM\_B (C++ enumerator), 438

kSE05x\_ECCurveParam\_PARAM\_G (C++ enumerator), 438

kSE05x\_ECCurveParam\_PARAM\_N (C++ enumerator), 438

kSE05x\_ECCurveParam\_PARAM\_PRIME (C++ enumerator), 438

kSE05x\_ECDAASignatureAlgo\_ECDA (C++ enumerator), 438

kSE05x\_ECDAASignatureAlgo\_NA (C++ enumerator), 438

kSE05x\_ECDAASignatureAlgo\_SHA (C++ enumerator), 439

kSE05x\_ECDAASignatureAlgo\_SHA\_224 (C++ enumerator), 439

kSE05x\_ECDAASignatureAlgo\_SHA\_256 (C++ enumerator), 439

kSE05x\_ECDAASignatureAlgo\_SHA\_384 (C++ enumerator), 439

kSE05x\_ECDAASignatureAlgo\_SHA\_512 (C++ enumerator), 439

kSE05x\_EDSignatureAlgo\_ED25519PH\_SHA\_512 (C++ enumerator), 439

kSE05x\_EDSignatureAlgo\_NA (C++ enumerator), 439

kSE05x\_GP\_TAG\_AID (C++ enumerator), 458

kSE05x\_GP\_TAG\_CONTRL\_REF\_PARM (C++ enumerator), 458

kSE05x\_GP\_TAG\_DR\_SE (C++ enumerator), 458

kSE05x\_GP\_TAG\_GET\_DATA (C++ enumerator), 458

kSE05x\_GP\_TAG\_KEY\_LEN (C++ enumerator), 458

kSE05x\_GP\_TAG\_KEY\_TYPE (C++ enumerator), 458

kSE05x\_GP\_TAG\_RECEIPT (C++ enumerator), 458

kSE05x\_GP\_TAG\_SCP\_PARAMS (C++ enumerator), 458

kSE05x\_HealthCheckMode\_CODE\_SIGNATURE (C++ enumerator), 439

kSE05x\_HealthCheckMode\_DYNAMIC\_FLASH\_INTEGRITY (C++ enumerator), 440

kSE05x\_HealthCheckMode\_FIPS (C++ enumerator), 439

kSE05x\_HealthCheckMode\_NA (C++ enumerator), 439

kSE05x\_HealthCheckMode\_SENSOR (C++ enumerator), 440

kSE05x\_HealthCheckMode\_SFR\_CHECK (C++ enumerator), 440

kSE05x\_HealthCheckMode\_SHIELDING (C++ enumerator), 440

kSE05x\_HkdfMode\_ExpandOnly (C++ enumerator), 440

kSE05x\_HkdfMode\_ExtractExpand (C++ enumerator), 440

kSE05x\_HkdfMode\_NA (C++ enumerator), 440

kSE05x\_I2CM\_Baud\_Rate\_100Khz (C++ enumerator), 440

kSE05x\_I2CM\_Baud\_Rate\_400Khz (C++ enumerator), 440

kSE05x\_I2CM\_Configure (C++ enumerator), 442

kSE05x\_I2CM\_I2C\_Config (C++ enumerator), 441

[kSE05x\\_I2CM\\_I2C\\_Nack\\_Fail \(C++ enumerator\), 441](#)  
[kSE05x\\_I2CM\\_I2C\\_Read\\_Error \(C++ enumerator\), 441](#)  
[kSE05x\\_I2CM\\_I2C\\_Time\\_Out\\_Error \(C++ enumerator\), 441](#)  
[kSE05x\\_I2CM\\_I2C\\_Write\\_Error \(C++ enumerator\), 441](#)  
[kSE05x\\_I2CM\\_Invalid\\_Length \(C++ enumerator\), 441](#)  
[kSE05x\\_I2CM\\_Invalid\\_Length\\_Encode \(C++ enumerator\), 441](#)  
[kSE05x\\_I2CM\\_Invalid\\_Tag \(C++ enumerator\), 441](#)  
[kSE05x\\_I2CM\\_None \(C++ enumerator\), 442](#)  
[kSE05x\\_I2CM\\_Read \(C++ enumerator\), 442](#)  
[kSE05x\\_I2CM\\_StructuralIssue \(C++ enumerator\), 442](#)  
[kSE05x\\_I2CM\\_Success \(C++ enumerator\), 441](#)  
[kSE05x\\_I2CM\\_Write \(C++ enumerator\), 442](#)  
[kSE05x\\_INS\\_ATTEST \(C++ enumerator\), 443](#)  
[kSE05x\\_INS\\_AUTH\\_OBJECT \(C++ enumerator\), 443](#)  
[kSE05x\\_INS\\_CRYPT \(C++ enumerator\), 443](#)  
[kSE05x\\_INS\\_I2CM\\_Attestation \(C macro\), 636](#)  
[kSE05x\\_INS\\_MASK\\_INS\\_CHAR \(C++ enumerator\), 442](#)  
[kSE05x\\_INS\\_MASK\\_INSTRUCTION \(C++ enumerator\), 442](#)  
[kSE05x\\_INS\\_MGMT \(C++ enumerator\), 443](#)  
[kSE05x\\_INS\\_NA \(C++ enumerator\), 442](#)  
[kSE05x\\_INS\\_PROCESS \(C++ enumerator\), 443](#)  
[kSE05x\\_INS\\_READ \(C++ enumerator\), 443](#)  
[kSE05x\\_INS\\_READ\\_With\\_Attestation \(C macro\), 637](#)  
[kSE05x\\_INS\\_TRANSIENT \(C++ enumerator\), 443](#)  
[kSE05x\\_INS\\_WRITE \(C++ enumerator\), 443](#)  
[kSE05x\\_KeyPart\\_NA \(C++ enumerator\), 443](#)  
[kSE05x\\_KeyPart\\_Pair \(C++ enumerator\), 443](#)  
[kSE05x\\_KeyPart\\_Private \(C++ enumerator\), 443](#)  
[kSE05x\\_KeyPart\\_Public \(C++ enumerator\), 443](#)  
[kSE05x\\_LockIndicator\\_NA \(C++ enumerator\), 444](#)  
[kSE05x\\_LockIndicator\\_PERSISTENT\\_LOCK \(C++ enumerator\), 444](#)  
[kSE05x\\_LockIndicator\\_TRANSIENT\\_LOCK \(C++ enumerator\), 444](#)  
[kSE05x\\_LockState\\_LOCKED \(C++ enumerator\), 444](#)  
[kSE05x\\_LockState\\_NA \(C++ enumerator\), 444](#)  
[kSE05x\\_Mac\\_Oper\\_Generate \(C++ enumerator\), 444](#)  
[kSE05x\\_Mac\\_Oper\\_NA \(C++ enumerator\), 444](#)  
[kSE05x\\_Mac\\_Oper\\_Validate \(C++ enumerator\), 444](#)  
[kSE05x\\_MACAlgo\\_CMIC\\_128 \(C++ enumerator\), 445](#)  
[kSE05x\\_MACAlgo\\_HMAC\\_SHA1 \(C++ enumerator\), 445](#)  
[kSE05x\\_MACAlgo\\_HMAC\\_SHA256 \(C++ enumerator\), 445](#)  
[kSE05x\\_MACAlgo\\_HMAC\\_SHA384 \(C++ enumerator\), 445](#)  
[kSE05x\\_MACAlgo\\_HMAC\\_SHA512 \(C++ enumerator\), 445](#)  
[kSE05x\\_MACAlgo\\_NA \(C++ enumerator\), 445](#)  
[kSE05x\\_MemoryType\\_NA \(C++ enumerator\), 445](#)  
[kSE05x\\_MemoryType\\_PERSISTENT \(C++ enumerator\), 445](#)  
[kSE05x\\_MemoryType\\_TRANSIENT\\_DESELECT \(C++ enumerator\), 445](#)  
[kSE05x\\_MemoryType\\_TRANSIENT\\_RESET \(C++ enumerator\), 445](#)  
[kSE05x\\_MemTyp\\_PERSISTENT \(C++ enumerator\), 446](#)  
[kSE05x\\_MemTyp\\_TRANSIENT\\_DESELECT \(C++ enumerator\), 446](#)  
[kSE05x\\_MemTyp\\_TRANSIENT\\_RESET \(C++ enumerator\), 446](#)  
[kSE05x\\_MoreIndicator\\_MORE \(C++ enumerator\), 446](#)  
[kSE05x\\_MoreIndicator\\_NA \(C++ enumerator\), 446](#)  
[kSE05x\\_MoreIndicator\\_NO\\_MORE \(C++ enumerator\), 446](#)  
[kSE05x\\_Origin\\_EXTERNAL \(C++ enumerator\), 447](#)  
[kSE05x\\_Origin\\_INTERNAL \(C++ enumerator\), 447](#)  
[kSE05x\\_Origin\\_NA \(C++ enumerator\), 447](#)  
[kSE05x\\_Origin\\_PROVISIONED \(C++ enumerator\), 447](#)  
[kSE05x\\_P1\\_AEAD \(C++ enumerator\), 448](#)  
[kSE05x\\_P1\\_AEAD\\_SP800\\_38D \(C++ enumerator\), 448](#)  
[kSE05x\\_P1\\_AES \(C++ enumerator\), 447](#)  
[kSE05x\\_P1\\_BINARY \(C++ enumerator\), 448](#)  
[kSE05x\\_P1\\_CIPHER \(C++ enumerator\), 448](#)  
[kSE05x\\_P1\\_COUNTER \(C++ enumerator\), 448](#)  
[kSE05x\\_P1\\_CRYPT\\_OBJ \(C++ enumerator\), 448](#)  
[kSE05x\\_P1\\_CURVE \(C++ enumerator\), 448](#)  
[kSE05x\\_P1\\_DEFAULT \(C++ enumerator\), 447](#)  
[kSE05x\\_P1\\_DES \(C++ enumerator\), 447](#)  
[kSE05x\\_P1\\_EC \(C++ enumerator\), 447](#)  
[kSE05x\\_P1\\_HMAC \(C++ enumerator\), 447](#)  
[kSE05x\\_P1\\_KEY\\_PAIR \(C++ enumerator\), 447](#)  
[kSE05x\\_P1\\_MAC \(C++ enumerator\), 448](#)  
[kSE05x\\_P1\\_MASK\\_CRED\\_TYPE \(C++ enumerator\), 447](#)  
[kSE05x\\_P1\\_MASK\\_KEY\\_TYPE \(C++ enumerator\), 447](#)

kSE05x\_P1\_NA (C++ *enumerator*), 447  
 kSE05x\_P1\_PCR (C++ *enumerator*), 448  
 kSE05x\_P1\_PRIVATE (C++ *enumerator*), 447  
 kSE05x\_P1\_PUBLIC (C++ *enumerator*), 447  
 kSE05x\_P1\_RSA (C++ *enumerator*), 447  
 kSE05x\_P1\_SIGNATURE (C++ *enumerator*), 448  
 kSE05x\_P1\_TLS (C++ *enumerator*), 448  
 kSE05x\_P1\_UNUSED (C++ *enumerator*), 447  
 kSE05x\_P1\_UserID (C++ *enumerator*), 448  
 kSE05x\_P2\_ATTEST (C++ *enumerator*), 449  
 kSE05x\_P2\_ATTRIBUTES (C++ *enumerator*), 449  
 kSE05x\_P2\_AUTH\_FIRST\_PART1 (C++ *enumerator*), 450  
 kSE05x\_P2\_AUTH\_FIRST\_PART2 (C++ *enumerator*), 448  
 kSE05x\_P2\_AUTH\_NONFIRST\_PART1 (C++ *enumerator*), 450  
 kSE05x\_P2\_AUTH\_NONFIRST\_PART2 (C++ *enumerator*), 448  
 kSE05x\_P2\_CHANGE\_KEY\_PART1 (C++ *enumerator*), 449  
 kSE05x\_P2\_CHANGE\_KEY\_PART2 (C++ *enumerator*), 449  
 kSE05x\_P2\_CM\_COMMAND (C++ *enumerator*), 450  
 kSE05x\_P2\_CPLC (C++ *enumerator*), 449  
 kSE05x\_P2\_CREATE (C++ *enumerator*), 448  
 kSE05x\_P2\_CRYPTOLIST (C++ *enumerator*), 450  
 kSE05x\_P2\_CURVE\_LIST (C++ *enumerator*), 449  
 kSE05x\_P2\_DECRYPT (C++ *enumerator*), 450  
 kSE05x\_P2\_DECRYPT\_ONESHOT (C++ *enumerator*), 449  
 kSE05x\_P2\_DEFAULT (C++ *enumerator*), 448  
 kSE05x\_P2\_DELETE\_ALL (C++ *enumerator*), 449  
 kSE05x\_P2\_DELETE\_CURVE (C++ *enumerator*), 450  
 kSE05x\_P2\_DELETE\_OBJECT (C++ *enumerator*), 449  
 kSE05x\_P2\_DH (C++ *enumerator*), 448  
 kSE05x\_P2\_DH\_REVERSE (C++ *enumerator*), 450  
 kSE05x\_P2\_DIVERSIFY (C++ *enumerator*), 448  
 kSE05x\_P2\_DUMP\_KEY (C++ *enumerator*), 449  
 kSE05x\_P2\_ENCRYPT (C++ *enumerator*), 450  
 kSE05x\_P2\_ENCRYPT\_ONESHOT (C++ *enumerator*), 449  
 kSE05x\_P2\_EXIST (C++ *enumerator*), 449  
 kSE05x\_P2\_EXPORT (C++ *enumerator*), 449  
 kSE05x\_P2\_FINAL (C++ *enumerator*), 448  
 kSE05x\_P2\_GENERATE (C++ *enumerator*), 448  
 kSE05x\_P2\_GENERATE\_ONESHOT (C++ *enumerator*), 450  
 kSE05x\_P2\_HKDF (C++ *enumerator*), 449  
 kSE05x\_P2\_HKDF\_EXPAND\_ONLY (C++ *enumerator*), 449  
 kSE05x\_P2\_I2CM (C++ *enumerator*), 449  
 kSE05x\_P2\_I2CM\_ATTESTED (C++ *enumerator*), 449  
 kSE05x\_P2\_ID (C++ *enumerator*), 449  
 kSE05x\_P2\_IMPORT (C++ *enumerator*), 449  
 kSE05x\_P2\_IMPORT\_EXT (C++ *enumerator*), 450  
 kSE05x\_P2\_INIT (C++ *enumerator*), 448  
 kSE05x\_P2\_KILL\_AUTH (C++ *enumerator*), 449  
 kSE05x\_P2\_LIST (C++ *enumerator*), 449  
 kSE05x\_P2\_MAC (C++ *enumerator*), 449  
 kSE05x\_P2\_MEMORY (C++ *enumerator*), 449  
 kSE05x\_P2\_MODE\_OF\_OPERATION (C++ *enumerator*), 450  
 kSE05x\_P2\_ONESHOT (C++ *enumerator*), 448  
 kSE05x\_P2\_PARAM (C++ *enumerator*), 450  
 kSE05x\_P2\_PBKDF (C++ *enumerator*), 449  
 kSE05x\_P2\_RANDOM (C++ *enumerator*), 450  
 kSE05x\_P2\_RAW (C++ *enumerator*), 450  
 kSE05x\_P2\_RESTRICT (C++ *enumerator*), 450  
 kSE05x\_P2\_SANITY (C++ *enumerator*), 450  
 kSE05x\_P2\_SCP (C++ *enumerator*), 450  
 kSE05x\_P2\_SESSION\_CLOSE (C++ *enumerator*), 449  
 kSE05x\_P2\_SESSION\_CREATE (C++ *enumerator*), 449  
 kSE05x\_P2\_SESSION\_POLICY (C++ *enumerator*), 449  
 kSE05x\_P2\_SESSION\_REFRESH (C++ *enumerator*), 449  
 kSE05x\_P2\_SESSION\_UserID (C++ *enumerator*), 449  
 kSE05x\_P2\_SIGN (C++ *enumerator*), 448  
 kSE05x\_P2\_SIGN\_ECDSA (C++ *enumerator*), 449  
 kSE05x\_P2\_SIZE (C++ *enumerator*), 448  
 kSE05x\_P2\_TIME (C++ *enumerator*), 449  
 kSE05x\_P2\_TLS\_PMS (C++ *enumerator*), 450  
 kSE05x\_P2\_TLS\_PRF\_CLI\_HELLO (C++ *enumerator*), 450  
 kSE05x\_P2\_TLS\_PRF\_CLI\_RND (C++ *enumerator*), 450  
 kSE05x\_P2\_TLS\_PRF\_SRV\_HELLO (C++ *enumerator*), 450  
 kSE05x\_P2\_TLS\_PRF\_SRV\_RND (C++ *enumerator*), 450  
 kSE05x\_P2\_TRANSPORT (C++ *enumerator*), 449  
 kSE05x\_P2\_TYPE (C++ *enumerator*), 449  
 kSE05x\_P2\_UNLOCK\_CHALLENGE (C++ *enumerator*), 449  
 kSE05x\_P2\_UPDATE (C++ *enumerator*), 448  
 kSE05x\_P2\_VALIDATE (C++ *enumerator*), 450  
 kSE05x\_P2\_VALIDATE\_ONESHOT (C++ *enumerator*), 450  
 kSE05x\_P2\_VARIANT (C++ *enumerator*), 449  
 kSE05x\_P2\_VERIFY (C++ *enumerator*), 448  
 kSE05x\_P2\_VERSION (C++ *enumerator*), 449



[kSE05x\\_PlatformSCPRequest\\_NA \(C++ enumerator\), 450](#)  
[kSE05x\\_PlatformSCPRequest\\_NOT\\_REQUIRED \(C++ enumerator\), 451](#)  
[kSE05x\\_PlatformSCPRequest\\_REQUIRED \(C++ enumerator\), 450](#)  
[kSE05x\\_RestrictMode\\_NA \(C++ enumerator\), 451](#)  
[kSE05x\\_RestrictMode\\_RESTRICT\\_ALL \(C++ enumerator\), 451](#)  
[kSE05x\\_RestrictMode\\_RESTRICT\\_NEW \(C++ enumerator\), 451](#)  
[kSE05x\\_Result\\_FAILURE \(C++ enumerator\), 451](#)  
[kSE05x\\_Result\\_NA \(C++ enumerator\), 451](#)  
[kSE05x\\_Result\\_SUCCESS \(C++ enumerator\), 451](#)  
[kSE05x\\_RSABitLength\\_1024 \(C++ enumerator\), 452](#)  
[kSE05x\\_RSABitLength\\_1152 \(C++ enumerator\), 452](#)  
[kSE05x\\_RSABitLength\\_2048 \(C++ enumerator\), 452](#)  
[kSE05x\\_RSABitLength\\_3072 \(C++ enumerator\), 452](#)  
[kSE05x\\_RSABitLength\\_4096 \(C++ enumerator\), 452](#)  
[kSE05x\\_RSABitLength\\_512 \(C++ enumerator\), 452](#)  
[kSE05x\\_RSABitLength\\_NA \(C++ enumerator\), 452](#)  
[kSE05x\\_RSAEncryptionAlgo\\_NA \(C++ enumerator\), 452](#)  
[kSE05x\\_RSAEncryptionAlgo\\_NO\\_PAD \(C++ enumerator\), 452](#)  
[kSE05x\\_RSAEncryptionAlgo\\_PKCS1 \(C++ enumerator\), 452](#)  
[kSE05x\\_RSAEncryptionAlgo\\_PKCS1\\_OAEP \(C++ enumerator\), 452](#)  
[kSE05x\\_RSAKeyComponent\\_DP \(C++ enumerator\), 453](#)  
[kSE05x\\_RSAKeyComponent\\_DQ \(C++ enumerator\), 453](#)  
[kSE05x\\_RSAKeyComponent\\_INVQ \(C++ enumerator\), 453](#)  
[kSE05x\\_RSAKeyComponent\\_MOD \(C++ enumerator\), 453](#)  
[kSE05x\\_RSAKeyComponent\\_NA \(C++ enumerator\), 453](#)  
[kSE05x\\_RSAKeyComponent\\_P \(C++ enumerator\), 453](#)  
[kSE05x\\_RSAKeyComponent\\_PRIV\\_EXP \(C++ enumerator\), 453](#)  
[kSE05x\\_RSAKeyComponent\\_PUB\\_EXP \(C++ enumerator\), 453](#)  
[kSE05x\\_RSAKeyComponent\\_Q \(C++ enumerator\), 453](#)  
[kSE05x\\_RSAKeyFormat\\_CRT \(C++ enumerator\), 453](#)  
[kSE05x\\_RSAKeyFormat\\_RAW \(C++ enumerator\), 453](#)  
[kSE05x\\_RSAPubKeyComp\\_MOD \(C++ enumerator\), 454](#)  
[kSE05x\\_RSAPubKeyComp\\_NA \(C++ enumerator\), 454](#)  
[kSE05x\\_RSAPubKeyComp\\_PUB\\_EXP \(C++ enumerator\), 454](#)  
[kSE05x\\_RSASignAlgo\\_NA \(C++ enumerator\), 454](#)  
[kSE05x\\_RSASignAlgo\\_SHA1\\_PKCS1\\_PSS \(C++ enumerator\), 454](#)  
[kSE05x\\_RSASignAlgo\\_SHA224\\_PKCS1\\_PSS \(C++ enumerator\), 454](#)  
[kSE05x\\_RSASignAlgo\\_SHA256\\_PKCS1\\_PSS \(C++ enumerator\), 454](#)  
[kSE05x\\_RSASignAlgo\\_SHA384\\_PKCS1\\_PSS \(C++ enumerator\), 454](#)  
[kSE05x\\_RSASignAlgo\\_SHA512\\_PKCS1\\_PSS \(C++ enumerator\), 454](#)  
[kSE05x\\_RSASignAlgo\\_SHA\\_224\\_PKCS1 \(C++ enumerator\), 454](#)  
[kSE05x\\_RSASignAlgo\\_SHA\\_256\\_PKCS1 \(C++ enumerator\), 454](#)  
[kSE05x\\_RSASignAlgo\\_SHA\\_384\\_PKCS1 \(C++ enumerator\), 454](#)  
[kSE05x\\_RSASignAlgo\\_SHA\\_512\\_PKCS1 \(C++ enumerator\), 454](#)  
[kSE05x\\_RSASignatureAlgo\\_NA \(C++ enumerator\), 455](#)  
[kSE05x\\_RSASignatureAlgo\\_SHA1\\_PKCS1 \(C++ enumerator\), 455](#)  
[kSE05x\\_RSASignatureAlgo\\_SHA1\\_PKCS1\\_PSS \(C++ enumerator\), 455](#)  
[kSE05x\\_RSASignatureAlgo\\_SHA224\\_PKCS1\\_PSS \(C++ enumerator\), 455](#)  
[kSE05x\\_RSASignatureAlgo\\_SHA256\\_PKCS1\\_PSS \(C++ enumerator\), 455](#)  
[kSE05x\\_RSASignatureAlgo\\_SHA384\\_PKCS1\\_PSS \(C++ enumerator\), 455](#)  
[kSE05x\\_RSASignatureAlgo\\_SHA512\\_PKCS1\\_PSS \(C++ enumerator\), 455](#)  
[kSE05x\\_RSASignatureAlgo\\_SHA\\_224\\_PKCS1 \(C++ enumerator\), 455](#)  
[kSE05x\\_RSASignatureAlgo\\_SHA\\_256\\_PKCS1 \(C++ enumerator\), 455](#)  
[kSE05x\\_RSASignatureAlgo\\_SHA\\_384\\_PKCS1 \(C++ enumerator\), 455](#)  
[kSE05x\\_RSASignatureAlgo\\_SHA\\_512\\_PKCS1 \(C++ enumerator\), 455](#)  
[kSE05x\\_SecObjTyp\\_AES\\_KEY \(C++ enumerator\), 456](#)  
[kSE05x\\_SecObjTyp\\_BINARY\\_FILE \(C++ enumerator\), 456](#)

[kSE05x\\_SecObjTyp\\_COUNTER \(C++ enumerator\), 456](#)  
[kSE05x\\_SecObjTyp\\_CURVE \(C++ enumerator\), 456](#)  
[kSE05x\\_SecObjTyp\\_DES\\_KEY \(C++ enumerator\), 456](#)  
[kSE05x\\_SecObjTyp\\_EC\\_KEY\\_PAIR \(C++ enumerator\), 456](#)  
[kSE05x\\_SecObjTyp\\_EC\\_PRIV\\_KEY \(C++ enumerator\), 456](#)  
[kSE05x\\_SecObjTyp\\_EC\\_PUB\\_KEY \(C++ enumerator\), 456](#)  
[kSE05x\\_SecObjTyp\\_HMAC\\_KEY \(C++ enumerator\), 456](#)  
[kSE05x\\_SecObjTyp\\_PCR \(C++ enumerator\), 456](#)  
[kSE05x\\_SecObjTyp\\_RSA\\_KEY\\_PAIR \(C++ enumerator\), 456](#)  
[kSE05x\\_SecObjTyp\\_RSA\\_KEY\\_PAIR\\_CRT \(C++ enumerator\), 456](#)  
[kSE05x\\_SecObjTyp\\_RSA\\_PRIV\\_KEY \(C++ enumerator\), 456](#)  
[kSE05x\\_SecObjTyp\\_RSA\\_PRIV\\_KEY\\_CRT \(C++ enumerator\), 456](#)  
[kSE05x\\_SecObjTyp\\_RSA\\_PUB\\_KEY \(C++ enumerator\), 456](#)  
[kSE05x\\_SecObjTyp\\_UserID \(C++ enumerator\), 456](#)  
[kSE05x\\_Security\\_None \(C++ enumerator\), 441](#)  
[kSE05x\\_SetIndicator\\_NA \(C++ enumerator\), 456](#)  
[kSE05x\\_SetIndicator\\_NOT\\_SET \(C++ enumerator\), 456](#)  
[kSE05x\\_SetIndicator\\_SET \(C++ enumerator\), 456](#)  
[kSE05x\\_Sign\\_Enc\\_Request \(C++ enumerator\), 441](#)  
[kSE05x\\_Sign\\_Request \(C++ enumerator\), 441](#)  
[kSE05x\\_SW12\\_COMMAND\\_NOT\\_ALLOWED \(C++ enumerator\), 457](#)  
[kSE05x\\_SW12\\_CONDITIONS\\_NOT\\_SATISFIED \(C++ enumerator\), 457](#)  
[kSE05x\\_SW12\\_DATA\\_INVALID \(C++ enumerator\), 457](#)  
[kSE05x\\_SW12\\_NA \(C++ enumerator\), 457](#)  
[kSE05x\\_SW12\\_NO\\_ERROR \(C++ enumerator\), 457](#)  
[kSE05x\\_SW12\\_SECURITY\\_STATUS \(C++ enumerator\), 457](#)  
[kSE05x\\_SW12\\_WRONG\\_DATA \(C++ enumerator\), 457](#)  
[kSE05x\\_SymmKeyType\\_AES \(C++ enumerator\), 457](#)  
[kSE05x\\_SymmKeyType\\_CMAC \(C++ enumerator\), 457](#)  
[kSE05x\\_SymmKeyType\\_DES \(C++ enumerator\), 457](#)  
[kSE05x\\_SymmKeyType\\_HMAC \(C++ enumerator\), 457](#)  
[kSE05x\\_TAG\\_1 \(C++ enumerator\), 458](#)  
[kSE05x\\_TAG\\_10 \(C++ enumerator\), 458](#)  
[kSE05x\\_TAG\\_2 \(C++ enumerator\), 458](#)  
[kSE05x\\_TAG\\_3 \(C++ enumerator\), 458](#)  
[kSE05x\\_TAG\\_4 \(C++ enumerator\), 458](#)  
[kSE05x\\_TAG\\_5 \(C++ enumerator\), 458](#)  
[kSE05x\\_TAG\\_6 \(C++ enumerator\), 458](#)  
[kSE05x\\_TAG\\_7 \(C++ enumerator\), 458](#)  
[kSE05x\\_TAG\\_8 \(C++ enumerator\), 458](#)  
[kSE05x\\_TAG\\_9 \(C++ enumerator\), 458](#)  
[kSE05x\\_TAG\\_I2CM\\_Config \(C++ enumerator\), 442](#)  
[kSE05x\\_TAG\\_I2CM\\_Read \(C++ enumerator\), 442](#)  
[kSE05x\\_TAG\\_I2CM\\_Write \(C++ enumerator\), 442](#)  
[kSE05x\\_TAG\\_IMPORT\\_AUTH\\_DATA \(C++ enumerator\), 458](#)  
[kSE05x\\_TAG\\_IMPORT\\_AUTH\\_KEY\\_ID \(C++ enumerator\), 458](#)  
[kSE05x\\_TAG\\_MAX\\_ATTEMPTS \(C++ enumerator\), 458](#)  
[kSE05x\\_TAG\\_NA \(C++ enumerator\), 458](#)  
[kSE05x\\_TAG\\_POLICY \(C++ enumerator\), 458](#)  
[kSE05x\\_TAG\\_SESSION\\_ID \(C++ enumerator\), 458](#)  
[kSE05x\\_TLS\\_PRF\\_CLI\\_HELLO \(C++ enumerator\), 459](#)  
[kSE05x\\_TLS\\_PRF\\_CLI\\_RND \(C++ enumerator\), 459](#)  
[kSE05x\\_TLS\\_PRF\\_NA \(C++ enumerator\), 459](#)  
[kSE05x\\_TLS\\_PRF\\_SRV\\_HELLO \(C++ enumerator\), 459](#)  
[kSE05x\\_TLS\\_PRF\\_SRV\\_RND \(C++ enumerator\), 459](#)  
[kSE05x\\_TransientIndicator\\_NA \(C++ enumerator\), 459](#)  
[kSE05x\\_TransientIndicator\\_PERSISTENT \(C++ enumerator\), 459](#)  
[kSE05x\\_TransientIndicator\\_TRANSIENT \(C++ enumerator\), 459](#)  
[kSE05x\\_TransientType\\_Persistent \(C++ enumerator\), 459](#)  
[kSE05x\\_TransientType\\_Transient \(C++ enumerator\), 459](#)  
[kSSS\\_AuthType\\_AESKey \(C++ enumerator\), 460](#)  
[kSSS\\_AuthType\\_AppletSCP03 \(C macro\), 637](#)  
[kSSS\\_AuthType\\_ECKKey \(C++ enumerator\), 460](#)  
[kSSS\\_AuthType\\_FastSCP \(C macro\), 637](#)  
[kSSS\\_AuthType\\_FastSCP\\_Counter \(C macro\), 637](#)  
[kSSS\\_AuthType\\_ID \(C++ enumerator\), 460](#)  
[kSSS\\_AuthType\\_INT\\_ECKKey\\_Counter \(C++ enumerator\), 460](#)  
[kSSS\\_AuthType\\_INT\\_FastSCP\\_Counter \(C macro\), 638](#)  
[kSSS\\_AuthType\\_None \(C++ enumerator\), 460](#)  
[kSSS\\_AuthType\\_SCP03 \(C++ enumerator\), 460](#)  
[kSSS\\_CipherType\\_AES \(C++ enumerator\), 463](#)  
[kSSS\\_CipherType\\_Binary \(C++ enumerator\), 463](#)  
[kSSS\\_CipherType\\_Certificate \(C++ enumerator\), 463](#)

[kSSS\\_CipherType\\_CMAL \(C++ enumerator\), 463](#)  
[kSSS\\_CipherType\\_Count \(C++ enumerator\), 463](#)  
[kSSS\\_CipherType\\_DES \(C++ enumerator\), 463](#)  
[kSSS\\_CipherType\\_EC\\_BARRETO\\_NAEHRIG \(C++ enumerator\), 463](#)  
[kSSS\\_CipherType\\_EC\\_BRAINPOOL \(C++ enumerator\), 463](#)  
[kSSS\\_CipherType\\_EC\\_MONTGOMERY \(C++ enumerator\), 463](#)  
[kSSS\\_CipherType\\_EC\\_NIST\\_K \(C++ enumerator\), 463](#)  
[kSSS\\_CipherType\\_EC\\_NIST\\_P \(C++ enumerator\), 463](#)  
[kSSS\\_CipherType\\_EC\\_TWISTED\\_ED \(C++ enumerator\), 463](#)  
[kSSS\\_CipherType\\_HMAC \(C++ enumerator\), 463](#)  
[kSSS\\_CipherType\\_MAC \(C++ enumerator\), 463](#)  
[kSSS\\_CipherType\\_NONE \(C++ enumerator\), 463](#)  
[kSSS\\_CipherType\\_PCR \(C++ enumerator\), 463](#)  
[kSSS\\_CipherType\\_ReservedPin \(C++ enumerator\), 463](#)  
[kSSS\\_CipherType\\_RSA \(C++ enumerator\), 463](#)  
[kSSS\\_CipherType\\_RSA\\_CRT \(C++ enumerator\), 463](#)  
[kSSS\\_CipherType\\_UserID \(C++ enumerator\), 463](#)  
[kSSS\\_ConnectionType\\_Encrypted \(C++ enumerator\), 464](#)  
[kSSS\\_ConnectionType\\_Password \(C++ enumerator\), 464](#)  
[kSSS\\_ConnectionType\\_Plain \(C++ enumerator\), 464](#)  
[kSSS\\_KeyPart\\_Default \(C++ enumerator\), 464](#)  
[kSSS\\_KeyPart\\_NONE \(C++ enumerator\), 464](#)  
[kSSS\\_KeyPart\\_Pair \(C++ enumerator\), 465](#)  
[kSSS\\_KeyPart\\_Private \(C++ enumerator\), 464](#)  
[kSSS\\_KeyPart\\_Public \(C++ enumerator\), 464](#)  
[kSSS\\_KeyStoreProp\\_au8\\_Optional\\_Start \(C++ enumerator\), 465](#)  
[kSSS\\_KeyStoreProp\\_FreeMem\\_Persistent \(C++ enumerator\), 468](#)  
[kSSS\\_KeyStoreProp\\_FreeMem\\_Transient \(C++ enumerator\), 468](#)  
[kSSS\\_SE05x\\_SessionProp\\_CertUID \(C++ enumerator\), 466](#)  
[kSSS\\_SE05x\\_SessionProp\\_CertUIDLen \(C++ enumerator\), 467](#)  
[kSSS\\_SessionProp\\_au8\\_NA \(C++ enumerator\), 467](#)  
[kSSS\\_SessionProp\\_au8\\_Optional\\_Start \(C++ enumerator\), 467](#)  
[kSSS\\_SessionProp\\_au8\\_Proprietary\\_Start \(C++ enumerator\), 467](#)  
[kSSS\\_SessionProp\\_szName \(C++ enumerator\), 467](#)  
[kSSS\\_SessionProp\\_u32\\_NA \(C++ enumerator\), 468](#)  
[kSSS\\_SessionProp\\_u32\\_Optional\\_Start \(C++ enumerator\), 468](#)  
[kSSS\\_SessionProp\\_u32\\_Proprietary\\_Start \(C++ enumerator\), 468](#)  
[kSSS\\_SessionProp\\_UID \(C++ enumerator\), 467](#)  
[kSSS\\_SessionProp\\_UIDLen \(C++ enumerator\), 468](#)  
[kSSS\\_SessionProp\\_VerDev \(C++ enumerator\), 468](#)  
[kSSS\\_SessionProp\\_VerMaj \(C++ enumerator\), 468](#)  
[kSSS\\_SessionProp\\_VerMin \(C++ enumerator\), 468](#)  
[kSSS\\_SIZE \(C++ enumerator\), 460](#)  
[kSSS\\_TunnelDest\\_None \(C++ enumerator\), 469](#)  
[kSSS\\_TunnelType\\_Se05x\\_Iot\\_applet \(C++ enumerator\), 469](#)  
[kStatus\\_SSS\\_Fail \(C++ enumerator\), 468](#)  
[kStatus\\_SSS\\_InvalidArgument \(C++ enumerator\), 468](#)  
[kStatus\\_SSS\\_ResourceBusy \(C++ enumerator\), 468](#)  
[kStatus\\_SSS\\_Success \(C++ enumerator\), 468](#)  
[kType\\_SSS\\_HW \(C++ enumerator\), 469](#)  
[kType\\_SSS\\_Isolated\\_HW \(C++ enumerator\), 469](#)  
[kType\\_SSS\\_mbedtlsTLS \(C++ enumerator\), 469](#)  
[kType\\_SSS\\_OpenSSL \(C++ enumerator\), 469](#)  
[kType\\_SSS\\_SE\\_A71CH \(C++ enumerator\), 469](#)  
[kType\\_SSS\\_SE\\_A71CL \(C++ enumerator\), 469](#)  
[kType\\_SSS\\_SE\\_SE05x \(C++ enumerator\), 470](#)  
[kType\\_SSS\\_SECO \(C++ enumerator\), 469](#)  
[kType\\_SSS\\_SecureElement \(C++ enumerator\), 469](#)  
[kType\\_SSS\\_Sentinel \(C++ enumerator\), 469](#)  
[kType\\_SSS\\_Sentinel200 \(C++ enumerator\), 469](#)  
[kType\\_SSS\\_Sentinel300 \(C++ enumerator\), 469](#)  
[kType\\_SSS\\_Sentinel400 \(C++ enumerator\), 469](#)  
[kType\\_SSS\\_Sentinel500 \(C++ enumerator\), 469](#)  
[kType\\_SSS\\_Software \(C++ enumerator\), 469](#)  
[kType\\_SSS\\_SubSystem\\_LAST \(C++ enumerator\), 470](#)  
[kType\\_SSS\\_SubSystem\\_NONE \(C++ enumerator\), 469](#)

**L**

[Log](#)  
[command line option, 113](#)

**M**

[main \(C++ function\), 471](#)  
[mbedtlsTLS\\_ALT](#)  
[command line option, 112](#)



mbedtls\_ecp\_keypair\_free\_o (C++ *function*),  
     471  
 mbedtls\_ecp\_tls\_read\_group\_o (C++ *func-*  
     *tion*), 472

## N

NX\_ENSURE\_DO\_LOG\_MESSAGE (C *macro*), 638  
 NX\_ENSURE\_MESSAGE (C *macro*), 638  
 NXECKKey03\_StaticCtx\_t (C++ *class*), 401  
 NXECKKey03\_StaticCtx\_t::HostEcdsaObj  
     (C++ *member*), 401  
 NXECKKey03\_StaticCtx\_t::HostEcKeypair  
     (C++ *member*), 401  
 NXECKKey03\_StaticCtx\_t::masterSec (C++  
     *member*), 401  
 NXECKKey03\_StaticCtx\_t::SeEcPubKey (C++  
     *member*), 401  
 NXPIInternal  
     command line option, 116  
 NXSCP03\_AuthCtx\_t (C++ *class*), 401  
 NXSCP03\_AuthCtx\_t::pDyn\_ctx (C++ *member*),  
     401  
 NXSCP03\_AuthCtx\_t::pStatic\_ctx (C++ *mem-*  
     *ber*), 401  
 NXSCP03\_DynCtx\_t (C++ *class*), 402  
 NXSCP03\_DynCtx\_t::authType (C++ *member*),  
     402  
 NXSCP03\_DynCtx\_t::cCounter (C++ *member*),  
     402  
 NXSCP03\_DynCtx\_t::Enc (C++ *member*), 402  
 NXSCP03\_DynCtx\_t::Mac (C++ *member*), 402  
 NXSCP03\_DynCtx\_t::MCV (C++ *member*), 402  
 NXSCP03\_DynCtx\_t::Rmac (C++ *member*), 402  
 NXSCP03\_DynCtx\_t::SecurityLevel (C++  
     *member*), 402  
 NXSCP03\_StaticCtx\_t (C++ *class*), 402  
 NXSCP03\_StaticCtx\_t::Dek (C++ *member*), 403  
 NXSCP03\_StaticCtx\_t::Enc (C++ *member*), 403  
 NXSCP03\_StaticCtx\_t::keyVerNo (C++ *mem-*  
     *ber*), 403  
 NXSCP03\_StaticCtx\_t::Mac (C++ *member*), 403

## P

PCONTEXT (C *macro*), 638

## R

RTOS  
     command line option, 112

## S

SCP  
     command line option, 112  
 SE050\_INS\_MASK\_INS\_CHAR (C *macro*), 639  
 SE050\_INS\_MASK\_INSTRUCTION (C *macro*), 639

SE050\_MAX\_APDU\_PAYLOAD\_LENGTH (C *macro*),  
     639  
 SE050\_MAX\_I2CM\_COMMAND\_LENGTH (C *macro*),  
     639  
 SE050\_MAX\_NUMBER\_OF\_SESSIONS (C *macro*), 639  
 SE050\_OBJECT\_IDENTIFIER\_SIZE (C *macro*), 640  
 SE05x\_AeadAlgo\_t (C++ *enum*), 430  
 Se05x\_API\_CheckObjectExists (C++ *function*),  
     472  
 Se05x\_API\_CipherFinal (C++ *function*), 473  
 Se05x\_API\_CipherInit (C++ *function*), 474  
 Se05x\_API\_CipherOneShot (C++ *function*), 475  
 Se05x\_API\_CipherUpdate (C++ *function*), 476  
 Se05x\_API\_CloseSession (C++ *function*), 477  
 Se05x\_API\_CreateCounter (C++ *function*), 478  
 Se05x\_API\_CreateCryptoObject (C++ *func-*  
     *tion*), 479  
 Se05x\_API\_CreateECCurve (C++ *function*), 480  
 Se05x\_API\_CreateSession (C++ *function*), 480  
 Se05x\_API\_DeleteAll (C++ *function*), 481  
 Se05x\_API\_DeleteAll\_Iterative (C++ *func-*  
     *tion*), 482  
 Se05x\_API\_DeleteCryptoObject (C++ *func-*  
     *tion*), 483  
 Se05x\_API\_DeleteECCurve (C++ *function*), 484  
 Se05x\_API\_DeleteSecureObject (C++ *func-*  
     *tion*), 484  
 Se05x\_API\_DFAuthenticateFirstPart1 (C++  
     *function*), 485  
 Se05x\_API\_DFAuthenticateFirstPart2 (C++  
     *function*), 486  
 Se05x\_API\_DFAuthenticateNonFirstPart1  
     (C++ *function*), 487  
 Se05x\_API\_DFChangeKeyPart1 (C++ *function*),  
     488  
 Se05x\_API\_DFChangeKeyPart2 (C++ *function*),  
     490  
 Se05x\_API\_DFDiversifyKey (C++ *function*), 491  
 Se05x\_API\_DFDumpSessionKeys (C++ *function*),  
     492  
 Se05x\_API\_DFKillAuthentication (C++ *func-*  
     *tion*), 493  
 Se05x\_API\_DigestFinal (C++ *function*), 494  
 Se05x\_API\_DigestInit (C++ *function*), 495  
 Se05x\_API\_DigestOneShot (C++ *function*), 495  
 Se05x\_API\_DigestUpdate (C++ *function*), 496  
 Se05x\_API\_EC\_CurveGetId (C++ *function*), 497  
 Se05x\_API\_ECDAASign (C++ *function*), 498  
 Se05x\_API\_ECDHGenerateSharedSecret (C++  
     *function*), 499  
 Se05x\_API\_ECDSASign (C++ *function*), 500  
 Se05x\_API\_ECDSAVerify (C++ *function*), 502  
 Se05x\_API\_ECGenSharedSecret (C *macro*), 640  
 Se05x\_API\_EdDSASign (C++ *function*), 503

- Se05x\_API\_EdDSAVerify (C++ function), 504
- Se05x\_API\_ExchangeSessionData (C++ function), 505
- Se05x\_API\_ExportObject (C++ function), 506
- Se05x\_API\_GetECCurveId (C++ function), 508
- Se05x\_API\_GetFreeMemory (C++ function), 508
- Se05x\_API\_GetRandom (C++ function), 509
- Se05x\_API\_GetTimestamp (C++ function), 510
- Se05x\_API\_GetVersion (C++ function), 511
- Se05x\_API\_HKDF (C++ function), 512
- Se05x\_API\_I2CM\_ExecuteCommandSet (C++ function), 514
- Se05x\_API\_ImportExternalObject (C++ function), 516
- Se05x\_API\_ImportObject (C++ function), 518
- Se05x\_API\_IncCounter (C++ function), 519
- Se05x\_API\_MACFinal (C++ function), 519
- Se05x\_API\_MACInit (C++ function), 521
- Se05x\_API\_MACOneShot\_G (C++ function), 522
- Se05x\_API\_MACOneShot\_V (C++ function), 523
- Se05x\_API\_MACUpdate (C++ function), 523
- Se05x\_API\_PBKDF2 (C++ function), 524
- Se05x\_API\_ReadCryptoObjectList (C++ function), 526
- Se05x\_API\_ReadECCurveList (C++ function), 527
- Se05x\_API\_ReadIDList (C++ function), 528
- Se05x\_API\_ReadObject (C++ function), 529
- Se05x\_API\_ReadObject\_W\_Attest (C++ function), 531
- Se05x\_API\_ReadObjectAttributes\_W\_Attest (C++ function), 533
- Se05x\_API\_ReadRSA (C++ function), 535
- Se05x\_API\_ReadRSA\_W\_Attest (C++ function), 536
- Se05x\_API\_ReadSize (C++ function), 537
- Se05x\_API\_ReadType (C++ function), 538
- Se05x\_API\_RefreshSession (C++ function), 539
- Se05x\_API\_RSADecrypt (C++ function), 540
- Se05x\_API\_RSAEncrypt (C++ function), 541
- Se05x\_API\_RSASign (C++ function), 542
- Se05x\_API\_RSAVerify (C++ function), 543
- Se05x\_API\_SetAppletFeatures (C++ function), 544
- Se05x\_API\_SetCounterValue (C++ function), 545
- Se05x\_API\_SetECCurveParam (C++ function), 545
- Se05x\_API\_SetLockState (C++ function), 546
- Se05x\_API\_SetPlatformSCPRequest (C++ function), 547
- Se05x\_API\_SHAOneShot (C macro), 640
- Se05x\_API\_TLSCalculatePreMasterSecret (C++ function), 548
- Se05x\_API\_TLSGenerateRandom (C++ function), 550
- Se05x\_API\_TLSPerformPRF (C++ function), 551
- Se05x\_API\_VerifySessionUserID (C++ function), 552
- Se05x\_API\_WriteBinary (C++ function), 553
- Se05x\_API\_WriteECKey (C++ function), 554
- Se05x\_API\_WritePCR (C++ function), 555
- Se05x\_API\_WriteRSAKey (C++ function), 556
- Se05x\_API\_WriteSymmKey (C++ function), 558
- Se05x\_API\_WriteUserID (C++ function), 559
- SE05x\_Applet\_Feature\_t (C++ class), 403
- SE05x\_Applet\_Feature\_t::AppletConfig\_AES (C++ member), 403
- SE05x\_Applet\_Feature\_t::AppletConfig\_DES (C++ member), 403
- SE05x\_Applet\_Feature\_t::AppletConfig\_DH\_MONT (C++ member), 403
- SE05x\_Applet\_Feature\_t::AppletConfig\_ECDSA (C++ member), 403
- SE05x\_Applet\_Feature\_t::AppletConfig\_ECDSA\_ECDH\_ECDHE (C++ member), 403
- SE05x\_Applet\_Feature\_t::AppletConfig\_EDDSA (C++ member), 403
- SE05x\_Applet\_Feature\_t::AppletConfig\_HMAC (C++ member), 403
- SE05x\_Applet\_Feature\_t::AppletConfig\_I2CM (C++ member), 403
- SE05x\_Applet\_Feature\_t::AppletConfig\_MIFARE (C++ member), 403
- SE05x\_Applet\_Feature\_t::AppletConfig\_PBKDF (C++ member), 403
- SE05x\_Applet\_Feature\_t::AppletConfig\_RFU1 (C++ member), 403
- SE05x\_Applet\_Feature\_t::AppletConfig\_RFU21 (C++ member), 403
- SE05x\_Applet\_Feature\_t::AppletConfig\_RSA\_CRT (C++ member), 404
- SE05x\_Applet\_Feature\_t::AppletConfig\_RSA\_PLAIN (C++ member), 404
- SE05x\_Applet\_Feature\_t::AppletConfig\_TLS (C++ member), 404
- SE05x\_AppletConfig\_t (C++ enum), 430
- SE05x\_AppletResID\_t (C++ enum), 431
- SE05x\_AttestationAlgo\_t (C++ enum), 432
- SE05x\_AttestationType\_t (C++ enum), 433
- SE05X\_Auth
  - command line option, 112
- se05x\_auth\_context\_t (C macro), 640
- SE05x\_AuthCtx\_ECKey\_t (C++ class), 404
- SE05x\_AuthCtx\_ECKey\_t::pDyn\_ctx (C++ member), 404
- SE05x\_AuthCtx\_ECKey\_t::pStatic\_ctx (C++ member), 404

SE05x\_AuthCtx\_ID\_t (C++ class), 404  
 SE05x\_AuthCtx\_ID\_t::pObj (C++ member), 405  
 SE05x\_AuthCtx\_t (C macro), 641  
 SE05x\_AuthType\_t (C macro), 641  
 SE05x\_Cipher\_Oper\_OneShot\_t (C++ enum), 433  
 SE05x\_Cipher\_Oper\_t (C++ enum), 434  
 SE05x\_CipherMode\_t (C++ enum), 434  
 SE05x\_Connect\_Ctx\_t (C macro), 641  
 SE05x\_CryptoContext\_t (C++ enum), 435  
 SE05x\_CryptoModeSubType\_t (C++ union), 470  
 SE05x\_CryptoModeSubType\_t::aead (C++ member), 470  
 SE05x\_CryptoModeSubType\_t::cipher (C++ member), 470  
 SE05x\_CryptoModeSubType\_t::digest (C++ member), 470  
 SE05x\_CryptoModeSubType\_t::mac (C++ member), 470  
 SE05x\_CryptoModeSubType\_t::union\_8bit (C++ member), 470  
 SE05x\_CryptoObject\_t (C++ enum), 435  
 SE05x\_CryptoObjectID\_t (C macro), 641  
 SE05x\_DigestMode\_t (C++ enum), 436  
 SE05x\_ECCurve\_t (C++ enum), 437  
 SE05x\_ECCurveParam\_t (C++ enum), 438  
 SE05x\_ECDAASignatureAlgo\_t (C++ enum), 438  
 SE05x\_ECSignatureAlgo\_t (C++ enum), 439  
 SE05x\_EDSignatureAlgo\_t (C++ enum), 439  
 se05x\_get\_sha\_algo (C++ function), 560  
 SE05x\_HealthCheckMode\_t (C++ enum), 439  
 SE05x\_HkdfMode\_t (C++ enum), 440  
 Se05x\_i2c\_master\_attst\_txn (C++ function), 560  
 Se05x\_i2c\_master\_txn (C++ function), 561  
 SE05x\_I2CM\_Baud\_Rate\_t (C++ enum), 440  
 SE05x\_I2CM\_cmd\_t (C++ class), 405  
 SE05x\_I2CM\_cmd\_t::cmd (C++ member), 405  
 SE05x\_I2CM\_cmd\_t::type (C++ member), 405  
 SE05x\_I2CM\_configData\_t (C++ class), 405  
 SE05x\_I2CM\_configData\_t::I2C\_addr (C++ member), 405  
 SE05x\_I2CM\_configData\_t::I2C\_baudRate (C++ member), 405  
 SE05x\_I2CM\_configData\_t::status (C++ member), 405  
 SE05x\_I2CM\_INS\_type\_t (C++ union), 470  
 SE05x\_I2CM\_INS\_type\_t::cfg (C++ member), 471  
 SE05x\_I2CM\_INS\_type\_t::issue (C++ member), 471  
 SE05x\_I2CM\_INS\_type\_t::rd (C++ member), 471  
 SE05x\_I2CM\_INS\_type\_t::sec (C++ member), 471  
 SE05x\_I2CM\_INS\_type\_t::w (C++ member), 471  
 SE05x\_I2CM\_readData\_t (C++ class), 406  
 SE05x\_I2CM\_readData\_t::rdBuf (C++ member), 406  
 SE05x\_I2CM\_readData\_t::rdStatus (C++ member), 406  
 SE05x\_I2CM\_readData\_t::readLength (C++ member), 406  
 SE05x\_I2CM\_securityData\_t (C++ class), 406  
 SE05x\_I2CM\_securityData\_t::keyObject (C++ member), 406  
 SE05x\_I2CM\_securityData\_t::operation (C++ member), 406  
 SE05x\_I2CM\_securityReq\_t (C++ enum), 441  
 SE05x\_I2CM\_status\_t (C++ enum), 441  
 SE05x\_I2CM\_structuralIssue\_t (C++ class), 407  
 SE05x\_I2CM\_structuralIssue\_t::issueStatus (C++ member), 407  
 SE05x\_I2CM\_TAG\_t (C++ enum), 442  
 SE05x\_I2CM\_TLV\_type\_t (C++ enum), 442  
 SE05x\_I2CM\_writeData\_t (C++ class), 407  
 SE05x\_I2CM\_writeData\_t::writebuf (C++ member), 407  
 SE05x\_I2CM\_writeData\_t::writeLength (C++ member), 407  
 SE05x\_I2CM\_writeData\_t::wrStatus (C++ member), 407  
 SE05x\_INS\_t (C++ enum), 442  
 Se05x\_IsInvalidRangeOfUID (C++ function), 562  
 SE05x\_KeyID\_KEK\_NONE (C macro), 641  
 SE05x\_KeyID\_MFDF\_NONE (C macro), 642  
 SE05x\_KeyID\_t (C++ type), 662  
 SE05x\_KeyPart\_t (C++ enum), 443  
 SE05x\_LockIndicator\_t (C++ enum), 444  
 SE05x\_LockState\_t (C++ enum), 444  
 SE05x\_Mac\_Oper\_t (C++ enum), 444  
 SE05x\_MACAlgo\_t (C++ enum), 445  
 SE05x\_MacOperation\_t (C++ type), 662  
 SE05x\_MaxAttempts\_NA (C macro), 642  
 SE05x\_MaxAttempts\_t (C++ type), 663  
 SE05x\_MaxAttempts\_UNLIMITED (C macro), 642  
 SE05x\_MemoryType\_t (C++ enum), 445  
 SE05x\_MemTyp\_t (C++ enum), 446  
 SE05x\_MoreIndicator\_t (C++ enum), 446  
 SE05x-Origin\_t (C++ enum), 447  
 SE05x\_P1\_t (C++ enum), 447  
 SE05x\_P2\_t (C++ enum), 448  
 SE05x\_PlatformSCPRequest\_t (C++ enum), 450  
 SE05x\_RestrictMode\_t (C++ enum), 451  
 SE05x\_Result\_t (C++ enum), 451

SE05x\_RSABitLength\_t (C++ *enum*), 452  
 SE05x\_RSAEncryptionAlgo\_t (C++ *enum*), 452  
 SE05x\_RSAKeyComponent\_t (C++ *enum*), 453  
 SE05x\_RSAKeyFormat\_t (C++ *enum*), 453  
 SE05x\_RSAPubKeyComp\_t (C++ *enum*), 454  
 SE05x\_RSASignAlgo\_t (C++ *enum*), 454  
 SE05x\_RSASignatureAlgo\_t (C++ *enum*), 455  
 SE05x\_SecObjTyp\_t (C++ *enum*), 456  
 SE05x\_SecureObjectType\_t (C++ *type*), 663  
 SE05x\_SetIndicator\_t (C++ *enum*), 456  
 se05x\_sssKeyTypeLenToCurveId (C++ *function*), 562  
 SE05x\_SW12\_t (C++ *enum*), 457  
 SE05x\_SymmKeyType\_t (C++ *enum*), 457  
 SE05x\_TAG\_t (C++ *enum*), 458  
 SE05x\_TLSPerformPRFType\_t (C++ *enum*), 459  
 SE05x\_TransientIndicator\_t (C++ *enum*), 459  
 SE05x\_TransientType\_t (C++ *enum*), 459  
 SE05x\_Variant\_t (C++ *type*), 663  
 SE05X\_Ver  
     command line option, 110  
 SE\_AuthCtx\_t (C++ *class*), 407  
 SE\_AuthCtx\_t::a71chAuthKeys (C++ *member*), 407  
 SE\_AuthCtx\_t::authType (C++ *member*), 407  
 SE\_AuthCtx\_t::ctx (C++ *member*), 407  
 SE\_AuthCtx\_t::data (C++ *member*), 408  
 SE\_AuthCtx\_t::eckey (C++ *member*), 408  
 SE\_AuthCtx\_t::extension (C++ *member*), 408  
 SE\_AuthCtx\_t::idobj (C++ *member*), 408  
 SE\_AuthCtx\_t::scp03 (C++ *member*), 408  
 SE\_AuthType\_t (C++ *enum*), 460  
 SE\_Connect\_Ctx\_t (C++ *class*), 408  
 SE\_Connect\_Ctx\_t::auth (C++ *member*), 408  
 SE\_Connect\_Ctx\_t::connType (C++ *member*), 408  
 SE\_Connect\_Ctx\_t::i2cAddress (C++ *member*), 408  
 SE\_Connect\_Ctx\_t::portName (C++ *member*), 408  
 SE\_Connect\_Ctx\_t::refresh\_session (C++ *member*), 408  
 SE\_Connect\_Ctx\_t::session\_policy (C++ *member*), 408  
 SE\_Connect\_Ctx\_t::sizeofStucture (C++ *member*), 408  
 SE\_Connect\_Ctx\_t::skip\_select\_applet (C++ *member*), 408  
 SE\_Connect\_Ctx\_t::tunnelCtx (C++ *member*), 409  
 SIMW\_INSTALL\_INC\_DIR  
     command line option, 116  
 SIMW\_INSTALL\_SHARE\_DIR  
     command line option, 116  
 SM\_SECURE\_SCP03\_KEYOBJ (C++ *class*), 409  
 SM\_SECURE\_SCP03\_KEYOBJ::pKeyDek (C++ *member*), 409  
 SM\_SECURE\_SCP03\_KEYOBJ::pKeyEnc (C++ *member*), 409  
 SM\_SECURE\_SCP03\_KEYOBJ::pKeyMac (C++ *member*), 409  
 SMCOM  
     command line option, 111  
 sss\_access\_permission\_t (C++ *enum*), 460  
 sss\_aead\_context\_free (C *macro*), 642  
 sss\_aead\_context\_free (C++ *function*), 562  
 sss\_aead\_context\_init (C *macro*), 643  
 sss\_aead\_context\_init (C++ *function*), 562  
 sss\_aead\_finish (C *macro*), 643  
 sss\_aead\_finish (C++ *function*), 563  
 sss\_aead\_init (C *macro*), 643  
 sss\_aead\_init (C++ *function*), 564  
 sss\_aead\_one\_go (C *macro*), 643  
 sss\_aead\_one\_go (C++ *function*), 564  
 sss\_aead\_t (C++ *class*), 409  
 sss\_aead\_t::algorithm (C++ *member*), 410  
 sss\_aead\_t::data (C++ *member*), 410  
 sss\_aead\_t::extension (C++ *member*), 410  
 sss\_aead\_t::keyObject (C++ *member*), 410  
 sss\_aead\_t::mode (C++ *member*), 410  
 sss\_aead\_t::session (C++ *member*), 410  
 SSS\_AEAD\_TYPE\_IS\_SE05X (C *macro*), 643  
 sss\_aead\_update (C *macro*), 644  
 sss\_aead\_update (C++ *function*), 565  
 sss\_aead\_update\_aad (C *macro*), 644  
 sss\_aead\_update\_aad (C++ *function*), 566  
 SSS\_AES\_BLOCK\_SIZE (C *macro*), 644  
 sss\_algorithm\_t (C++ *enum*), 461  
 SSS\_API\_VERSION (C *macro*), 644  
 sss\_asymmetric\_context\_free (C *macro*), 645  
 sss\_asymmetric\_context\_free (C++ *function*), 566  
 sss\_asymmetric\_context\_init (C *macro*), 645  
 sss\_asymmetric\_context\_init (C++ *function*), 567  
 sss\_asymmetric\_decrypt (C *macro*), 645  
 sss\_asymmetric\_decrypt (C++ *function*), 567  
 sss\_asymmetric\_encrypt (C *macro*), 645  
 sss\_asymmetric\_encrypt (C++ *function*), 568  
 sss\_asymmetric\_sign\_digest (C *macro*), 645  
 sss\_asymmetric\_sign\_digest (C++ *function*), 568  
 sss\_asymmetric\_t (C++ *class*), 410  
 sss\_asymmetric\_t::algorithm (C++ *member*), 410  
 sss\_asymmetric\_t::data (C++ *member*), 410  
 sss\_asymmetric\_t::extension (C++ *member*), 410

sss\_asymmetric\_t::keyObject (C++ member), 410  
 sss\_asymmetric\_t::mode (C++ member), 410  
 sss\_asymmetric\_t::session (C++ member), 410  
 SSS\_ASYMMETRIC\_TYPE\_IS\_SE05X (C macro), 646  
 sss\_asymmetric\_verify\_digest (C macro), 646  
 sss\_asymmetric\_verify\_digest (C++ function), 569  
 sss\_cipher\_crypt\_ctr (C macro), 646  
 sss\_cipher\_crypt\_ctr (C++ function), 569  
 sss\_cipher\_finish (C macro), 646  
 sss\_cipher\_finish (C++ function), 570  
 sss\_cipher\_init (C macro), 647  
 sss\_cipher\_init (C++ function), 571  
 sss\_cipher\_one\_go (C macro), 647  
 sss\_cipher\_one\_go (C++ function), 571  
 sss\_cipher\_type\_t (C++ enum), 463  
 sss\_cipher\_update (C macro), 647  
 sss\_cipher\_update (C++ function), 572  
 sss\_connect\_ctx\_t (C++ class), 411  
 sss\_connect\_ctx\_t::auth (C++ member), 411  
 sss\_connect\_ctx\_t::data (C++ member), 411  
 sss\_connect\_ctx\_t::extension (C++ member), 411  
 sss\_connect\_ctx\_t::session\_policy (C++ member), 411  
 sss\_connect\_ctx\_t::sizeofStructure (C++ member), 411  
 sss\_connection\_type\_t (C++ enum), 464  
 sss\_derive\_key\_context\_free (C macro), 647  
 sss\_derive\_key\_context\_free (C++ function), 572  
 sss\_derive\_key\_context\_init (C macro), 647  
 sss\_derive\_key\_context\_init (C++ function), 573  
 sss\_derive\_key\_dh (C macro), 648  
 sss\_derive\_key\_dh (C++ function), 573  
 sss\_derive\_key\_go (C macro), 648  
 sss\_derive\_key\_go (C++ function), 574  
 sss\_derive\_key\_one\_go (C++ function), 575  
 sss\_derive\_key\_sobj\_one\_go (C++ function), 575  
 sss\_derive\_key\_t (C++ class), 411  
 sss\_derive\_key\_t::algorithm (C++ member), 411  
 sss\_derive\_key\_t::data (C++ member), 411  
 sss\_derive\_key\_t::extension (C++ member), 411  
 sss\_derive\_key\_t::keyObject (C++ member), 411  
 sss\_derive\_key\_t::mode (C++ member), 411  
 sss\_derive\_key\_t::session (C++ member), 411  
 SSS\_DERIVE\_KEY\_TYPE\_IS\_SE05X (C macro), 648  
 SSS\_DES\_BLOCK\_SIZE (C macro), 648  
 SSS\_DES\_IV\_SIZE (C macro), 649  
 SSS\_DES\_KEY\_SIZE (C macro), 649  
 sss\_digest\_context\_free (C macro), 649  
 sss\_digest\_context\_free (C++ function), 576  
 sss\_digest\_context\_init (C macro), 649  
 sss\_digest\_context\_init (C++ function), 576  
 sss\_digest\_finish (C macro), 649  
 sss\_digest\_finish (C++ function), 577  
 sss\_digest\_init (C macro), 650  
 sss\_digest\_init (C++ function), 577  
 sss\_digest\_one\_go (C macro), 650  
 sss\_digest\_one\_go (C++ function), 578  
 sss\_digest\_t (C++ class), 412  
 sss\_digest\_t::algorithm (C++ member), 412  
 sss\_digest\_t::data (C++ member), 412  
 sss\_digest\_t::digestFullLen (C++ member), 412  
 sss\_digest\_t::extension (C++ member), 412  
 sss\_digest\_t::mode (C++ member), 412  
 sss\_digest\_t::session (C++ member), 412  
 SSS\_DIGEST\_TYPE\_IS\_SE05X (C macro), 650  
 sss\_digest\_update (C macro), 650  
 sss\_digest\_update (C++ function), 578  
 sss\_ecc\_point\_t (C++ class), 412  
 sss\_ecc\_point\_t::X (C++ member), 412  
 sss\_ecc\_point\_t::Y (C++ member), 412  
 sss\_eccgfp\_group\_t (C++ class), 413  
 sss\_eccgfp\_group\_t::a (C++ member), 413  
 sss\_eccgfp\_group\_t::b (C++ member), 413  
 sss\_eccgfp\_group\_t::G (C++ member), 413  
 sss\_eccgfp\_group\_t::h (C++ member), 413  
 sss\_eccgfp\_group\_t::n (C++ member), 413  
 sss\_eccgfp\_group\_t::p (C++ member), 413  
 SSS\_ENUM (C macro), 651  
 sss\_key\_object\_allocate\_handle (C macro), 651  
 sss\_key\_object\_allocate\_handle (C++ function), 579  
 sss\_key\_object\_free (C macro), 651  
 sss\_key\_object\_free (C++ function), 579  
 sss\_key\_object\_get\_access (C macro), 651  
 sss\_key\_object\_get\_access (C++ function), 580  
 sss\_key\_object\_get\_handle (C macro), 651  
 sss\_key\_object\_get\_handle (C++ function), 580  
 sss\_key\_object\_get\_purpose (C macro), 652  
 sss\_key\_object\_get\_purpose (C++ function), 580  
 sss\_key\_object\_get\_user (C macro), 652  
 sss\_key\_object\_get\_user (C++ function), 581  
 sss\_key\_object\_init (C macro), 652



`sss_key_object_init` (C++ function), 581  
`sss_key_object_mode_t` (C++ enum), 464  
`sss_key_object_set_access` (C macro), 652  
`sss_key_object_set_access` (C++ function), 581  
`sss_key_object_set_eccgfp_group` (C macro), 653  
`sss_key_object_set_eccgfp_group` (C++ function), 582  
`sss_key_object_set_purpose` (C macro), 653  
`sss_key_object_set_purpose` (C++ function), 582  
`sss_key_object_set_user` (C macro), 653  
`sss_key_object_set_user` (C++ function), 583  
`sss_key_part_t` (C++ enum), 464  
`sss_key_store_allocate` (C macro), 653  
`sss_key_store_allocate` (C++ function), 583  
`sss_key_store_context_free` (C macro), 653  
`sss_key_store_context_free` (C++ function), 583  
`sss_key_store_context_init` (C macro), 654  
`sss_key_store_context_init` (C++ function), 584  
`sss_key_store_erase_key` (C macro), 654  
`sss_key_store_erase_key` (C++ function), 584  
`sss_key_store_freeze_key` (C macro), 654  
`sss_key_store_freeze_key` (C++ function), 584  
`sss_key_store_generate_key` (C macro), 654  
`sss_key_store_generate_key` (C++ function), 585  
`sss_key_store_get_key` (C macro), 655  
`sss_key_store_get_key` (C++ function), 585  
`sss_key_store_load` (C macro), 655  
`sss_key_store_load` (C++ function), 585  
`sss_key_store_open_key` (C macro), 655  
`sss_key_store_open_key` (C++ function), 585  
`sss_key_store_prop_a8_t` (C++ enum), 465  
`sss_key_store_save` (C macro), 655  
`sss_key_store_save` (C++ function), 586  
`sss_key_store_set_key` (C macro), 655  
`sss_key_store_set_key` (C++ function), 586  
`sss_key_store_t` (C++ class), 413  
`sss_key_store_t::data` (C++ member), 413  
`sss_key_store_t::extension` (C++ member), 413  
`sss_key_store_t::session` (C++ member), 413  
`SSS_KEY_STORE_TYPE_IS_SE05X` (C macro), 656  
`sss_mac_context_free` (C macro), 656  
`sss_mac_context_free` (C++ function), 586  
`sss_mac_context_init` (C macro), 656  
`sss_mac_context_init` (C++ function), 587  
`sss_mac_finish` (C macro), 656  
`sss_mac_finish` (C++ function), 587  
`sss_mac_init` (C macro), 657  
`sss_mac_init` (C++ function), 588  
`sss_mac_one_go` (C macro), 657  
`sss_mac_one_go` (C++ function), 588  
`sss_mac_t` (C++ class), 414  
`sss_mac_t::algorithm` (C++ member), 414  
`sss_mac_t::data` (C++ member), 414  
`sss_mac_t::extension` (C++ member), 414  
`sss_mac_t::keyObject` (C++ member), 414  
`sss_mac_t::mode` (C++ member), 414  
`sss_mac_t::session` (C++ member), 414  
`SSS_MAC_TYPE_IS_SE05X` (C macro), 657  
`sss_mac_update` (C macro), 657  
`sss_mac_update` (C++ function), 589  
`sss_mbedtlsls_associate_ecdhctx` (C++ function), 589  
`sss_mbedtlsls_associate_keypair` (C++ function), 590  
`sss_mbedtlsls_associate_pubkey` (C++ function), 590  
`sss_mode_t` (C++ enum), 465  
`sss_object_t` (C++ class), 414  
`sss_object_t::cipherType` (C++ member), 414  
`sss_object_t::data` (C++ member), 414  
`sss_object_t::extension` (C++ member), 414  
`sss_object_t::keyId` (C++ member), 414  
`sss_object_t::keyStore` (C++ member), 414  
`sss_object_t::objectType` (C++ member), 414  
`SSS_OBJECT_TYPE_IS_SE05X` (C macro), 657  
`sss_policy_asym_key_u` (C++ class), 415  
`sss_policy_asym_key_u::can_Attest` (C++ member), 415  
`sss_policy_asym_key_u::can_Decrypt` (C++ member), 415  
`sss_policy_asym_key_u::can_Encrypt` (C++ member), 415  
`sss_policy_asym_key_u::can_Gen` (C++ member), 415  
`sss_policy_asym_key_u::can_Import_Export` (C++ member), 415  
`sss_policy_asym_key_u::can_KA` (C++ member), 415  
`sss_policy_asym_key_u::can_KD` (C++ member), 415  
`sss_policy_asym_key_u::can_Read` (C++ member), 415  
`sss_policy_asym_key_u::can_Sign` (C++ member), 415  
`sss_policy_asym_key_u::can_Verify` (C++ member), 415  
`sss_policy_asym_key_u::can_Wrap` (C++ member), 415  
`sss_policy_asym_key_u::can_Write` (C++ member), 415  
`sss_policy_asym_key_u::forbid_Derived_Output`

(C++ member), 415  
 sss\_policy\_common\_pcr\_value\_u (C++ class), 416  
 sss\_policy\_common\_pcr\_value\_u::pcrExpectedValue (C++ member), 416  
 sss\_policy\_common\_pcr\_value\_u::pcrObjId (C++ member), 416  
 sss\_policy\_common\_u (C++ class), 416  
 sss\_policy\_common\_u::can\_Delete (C++ member), 416  
 sss\_policy\_common\_u::forbid\_All (C++ member), 416  
 sss\_policy\_common\_u::req\_Sm (C++ member), 416  
 sss\_policy\_counter\_u (C++ class), 417  
 sss\_policy\_counter\_u::can\_Read (C++ member), 417  
 sss\_policy\_counter\_u::can\_Write (C++ member), 417  
 sss\_policy\_file\_u (C++ class), 417  
 sss\_policy\_file\_u::can\_Read (C++ member), 417  
 sss\_policy\_file\_u::can\_Write (C++ member), 417  
 sss\_policy\_pcr\_u (C++ class), 417  
 sss\_policy\_pcr\_u::can\_Read (C++ member), 418  
 sss\_policy\_pcr\_u::can\_Write (C++ member), 418  
 sss\_policy\_session\_u (C++ class), 418  
 sss\_policy\_session\_u::allowRefresh (C++ member), 418  
 sss\_policy\_session\_u::has\_MaxDurationOfSession (C++ member), 418  
 sss\_policy\_session\_u::has\_MaxOperationsInSession (C++ member), 418  
 sss\_policy\_session\_u::maxDurationOfSession (C++ member), 418  
 sss\_policy\_session\_u::maxOperationsInSession (C++ member), 418  
 sss\_policy\_sym\_key\_u (C++ class), 418  
 sss\_policy\_sym\_key\_u::can\_Decrypt (C++ member), 419  
 sss\_policy\_sym\_key\_u::can\_Desfire\_Auth (C++ member), 419  
 sss\_policy\_sym\_key\_u::can\_Desfire\_Dump (C++ member), 419  
 sss\_policy\_sym\_key\_u::can\_Encrypt (C++ member), 419  
 sss\_policy\_sym\_key\_u::can\_Gen (C++ member), 419  
 sss\_policy\_sym\_key\_u::can\_Import\_Export (C++ member), 419  
 sss\_policy\_sym\_key\_u::can\_KD (C++ member), 419  
 sss\_policy\_sym\_key\_u::can\_Sign (C++ member), 419  
 sss\_policy\_sym\_key\_u::can\_Verify (C++ member), 419  
 sss\_policy\_sym\_key\_u::can\_Wrap (C++ member), 419  
 sss\_policy\_sym\_key\_u::can\_Write (C++ member), 419  
 sss\_policy\_sym\_key\_u::forbid\_Derived\_Output (C++ member), 419  
 sss\_policy\_t (C++ class), 419  
 sss\_policy\_t::nPolicies (C++ member), 420  
 sss\_policy\_t::policies (C++ member), 420  
 sss\_policy\_type\_u (C++ enum), 466  
 sss\_policy\_u (C++ class), 420  
 sss\_policy\_u::asymmkey (C++ member), 420  
 sss\_policy\_u::auth\_obj\_id (C++ member), 420  
 sss\_policy\_u::common (C++ member), 420  
 sss\_policy\_u::common\_pcr\_value (C++ member), 420  
 sss\_policy\_u::counter (C++ member), 420  
 sss\_policy\_u::file (C++ member), 420  
 sss\_policy\_u::pcr (C++ member), 420  
 sss\_policy\_u::pin (C++ member), 420  
 sss\_policy\_u::policy (C++ member), 420  
 sss\_policy\_u::session (C++ member), 420  
 sss\_policy\_u::symmkey (C++ member), 420  
 sss\_policy\_u::type (C++ member), 420  
 sss\_policy\_userid\_u (C++ class), 421  
 sss\_policy\_userid\_u::can\_Write (C++ member), 421  
 sss\_rng\_context\_free (C macro), 658  
 sss\_rng\_context\_free (C++ function), 590  
 sss\_rng\_context\_init (C macro), 658  
 sss\_rng\_context\_init (C++ function), 591  
 sss\_rng\_context\_t (C++ class), 421  
 sss\_rng\_context\_t::context (C++ member), 421  
 sss\_rng\_context\_t::data (C++ member), 421  
 sss\_rng\_context\_t::session (C++ member), 421  
 SSS\_RNG\_CONTEXT\_TYPE\_IS\_SE05X (C macro), 658  
 sss\_rng\_get\_random (C macro), 658  
 sss\_rng\_get\_random (C++ function), 591  
 sss\_s05x\_sesion\_prop\_a8\_t (C++ enum), 466  
 sss\_s05x\_sesion\_prop\_u32\_t (C++ enum), 467  
 sss\_se05x\_aead\_context\_free (C++ function), 592  
 sss\_se05x\_aead\_context\_init (C++ function), 592  
 sss\_se05x\_aead\_finish (C++ function), 593

[sss\\_se05x\\_aead\\_init \(C++ function\), 593](#)  
[sss\\_se05x\\_aead\\_one\\_go \(C++ function\), 594](#)  
[sss\\_se05x\\_aead\\_t \(C++ class\), 421](#)  
[sss\\_se05x\\_aead\\_t::algorithm \(C++ member\), 421](#)  
[sss\\_se05x\\_aead\\_t::cache\\_data \(C++ member\), 421](#)  
[sss\\_se05x\\_aead\\_t::cache\\_data\\_len \(C++ member\), 421](#)  
[sss\\_se05x\\_aead\\_t::cryptoObjectId \(C++ member\), 421](#)  
[sss\\_se05x\\_aead\\_t::keyObject \(C++ member\), 422](#)  
[sss\\_se05x\\_aead\\_t::mode \(C++ member\), 422](#)  
[sss\\_se05x\\_aead\\_t::session \(C++ member\), 422](#)  
[sss\\_se05x\\_aead\\_update \(C++ function\), 595](#)  
[sss\\_se05x\\_aead\\_update\\_aad \(C++ function\), 595](#)  
[sss\\_se05x\\_asymmetric\\_context\\_free \(C++ function\), 596](#)  
[sss\\_se05x\\_asymmetric\\_context\\_init \(C++ function\), 596](#)  
[sss\\_se05x\\_asymmetric\\_decrypt \(C++ function\), 597](#)  
[sss\\_se05x\\_asymmetric\\_encrypt \(C++ function\), 597](#)  
[sss\\_se05x\\_asymmetric\\_sign \(C++ function\), 598](#)  
[sss\\_se05x\\_asymmetric\\_sign\\_digest \(C++ function\), 598](#)  
[sss\\_se05x\\_asymmetric\\_t \(C++ class\), 422](#)  
[sss\\_se05x\\_asymmetric\\_t::algorithm \(C++ member\), 422](#)  
[sss\\_se05x\\_asymmetric\\_t::keyObject \(C++ member\), 422](#)  
[sss\\_se05x\\_asymmetric\\_t::mode \(C++ member\), 422](#)  
[sss\\_se05x\\_asymmetric\\_t::session \(C++ member\), 422](#)  
[sss\\_se05x\\_asymmetric\\_verify \(C++ function\), 599](#)  
[sss\\_se05x\\_asymmetric\\_verify\\_digest \(C++ function\), 599](#)  
[sss\\_se05x\\_attst\\_comp\\_data\\_t \(C++ class\), 422](#)  
[sss\\_se05x\\_attst\\_comp\\_data\\_t::attribute \(C++ member\), 423](#)  
[sss\\_se05x\\_attst\\_comp\\_data\\_t::attributeLen \(C++ member\), 423](#)  
[sss\\_se05x\\_attst\\_comp\\_data\\_t::chipId \(C++ member\), 423](#)  
[sss\\_se05x\\_attst\\_comp\\_data\\_t::chipIdLen \(C++ member\), 423](#)  
[sss\\_se05x\\_attst\\_comp\\_data\\_t::outrandom \(C++ member\), 423](#)  
[sss\\_se05x\\_attst\\_comp\\_data\\_t::outrandomLen \(C++ member\), 423](#)  
[sss\\_se05x\\_attst\\_comp\\_data\\_t::signature \(C++ member\), 423](#)  
[sss\\_se05x\\_attst\\_comp\\_data\\_t::signatureLen \(C++ member\), 423](#)  
[sss\\_se05x\\_attst\\_comp\\_data\\_t::timeStamp \(C++ member\), 423](#)  
[sss\\_se05x\\_attst\\_comp\\_data\\_t::timeStampLen \(C++ member\), 423](#)  
[sss\\_se05x\\_attst\\_data\\_t \(C++ class\), 423](#)  
[sss\\_se05x\\_attst\\_data\\_t::data \(C++ member\), 423](#)  
[sss\\_se05x\\_attst\\_data\\_t::valid\\_number \(C++ member\), 423](#)  
[sss\\_se05x\\_cipher\\_crypt\\_ctr \(C++ function\), 600](#)  
[sss\\_se05x\\_cipher\\_finish \(C++ function\), 600](#)  
[sss\\_se05x\\_cipher\\_init \(C++ function\), 601](#)  
[sss\\_se05x\\_cipher\\_one\\_go \(C++ function\), 601](#)  
[sss\\_se05x\\_cipher\\_update \(C++ function\), 602](#)  
[sss\\_se05x\\_derive\\_key\\_context\\_free \(C++ function\), 602](#)  
[sss\\_se05x\\_derive\\_key\\_context\\_init \(C++ function\), 603](#)  
[sss\\_se05x\\_derive\\_key\\_dh \(C++ function\), 603](#)  
[sss\\_se05x\\_derive\\_key\\_go \(C++ function\), 604](#)  
[sss\\_se05x\\_derive\\_key\\_one\\_go \(C++ function\), 605](#)  
[sss\\_se05x\\_derive\\_key\\_sobj\\_one\\_go \(C++ function\), 605](#)  
[sss\\_se05x\\_derive\\_key\\_t \(C++ class\), 424](#)  
[sss\\_se05x\\_derive\\_key\\_t::algorithm \(C++ member\), 424](#)  
[sss\\_se05x\\_derive\\_key\\_t::keyObject \(C++ member\), 424](#)  
[sss\\_se05x\\_derive\\_key\\_t::mode \(C++ member\), 424](#)  
[sss\\_se05x\\_derive\\_key\\_t::session \(C++ member\), 424](#)  
[sss\\_se05x\\_digest\\_context\\_free \(C++ function\), 606](#)  
[sss\\_se05x\\_digest\\_context\\_init \(C++ function\), 606](#)  
[sss\\_se05x\\_digest\\_finish \(C++ function\), 607](#)  
[sss\\_se05x\\_digest\\_init \(C++ function\), 607](#)  
[sss\\_se05x\\_digest\\_one\\_go \(C++ function\), 608](#)  
[sss\\_se05x\\_digest\\_t \(C++ class\), 424](#)  
[sss\\_se05x\\_digest\\_t::algorithm \(C++ member\), 424](#)  
[sss\\_se05x\\_digest\\_t::cryptoObjectId \(C++ member\), 424](#)



sss\_se05x\_digest\_t::digestFullLen (C++ member), 424  
 sss\_se05x\_digest\_t::mode (C++ member), 424  
 sss\_se05x\_digest\_t::session (C++ member), 424  
 sss\_se05x\_digest\_update (C++ function), 608  
 sss\_se05x\_key\_object\_allocate\_handle (C++ function), 609  
 sss\_se05x\_key\_object\_free (C++ function), 609  
 sss\_se05x\_key\_object\_get\_access (C++ function), 610  
 sss\_se05x\_key\_object\_get\_handle (C++ function), 610  
 sss\_se05x\_key\_object\_get\_purpose (C++ function), 610  
 sss\_se05x\_key\_object\_get\_user (C++ function), 611  
 sss\_se05x\_key\_object\_init (C++ function), 611  
 sss\_se05x\_key\_object\_set\_access (C++ function), 611  
 sss\_se05x\_key\_object\_set\_eccgfp\_group (C++ function), 612  
 sss\_se05x\_key\_object\_set\_purpose (C++ function), 612  
 sss\_se05x\_key\_object\_set\_user (C++ function), 612  
 sss\_se05x\_key\_store\_allocate (C++ function), 613  
 sss\_se05x\_key\_store\_context\_free (C++ function), 613  
 sss\_se05x\_key\_store\_context\_init (C++ function), 613  
 sss\_se05x\_key\_store\_create\_curve (C++ function), 614  
 sss\_se05x\_key\_store\_erase\_key (C++ function), 614  
 sss\_se05x\_key\_store\_export\_key (C++ function), 614  
 sss\_se05x\_key\_store\_freeze\_key (C++ function), 614  
 sss\_se05x\_key\_store\_generate\_key (C++ function), 615  
 sss\_se05x\_key\_store\_get\_key (C++ function), 615  
 sss\_se05x\_key\_store\_get\_key\_attst (C++ function), 615  
 sss\_se05x\_key\_store\_import\_key (C++ function), 616  
 sss\_se05x\_key\_store\_load (C++ function), 616  
 sss\_se05x\_key\_store\_open\_key (C++ function), 616  
 sss\_se05x\_key\_store\_save (C++ function), 617  
 sss\_se05x\_key\_store\_set\_key (C++ function), 617  
 sss\_se05x\_key\_store\_t (C++ class), 425  
 sss\_se05x\_key\_store\_t::kekKey (C++ member), 425  
 sss\_se05x\_key\_store\_t::session (C++ member), 425  
 sss\_se05x\_mac\_context\_free (C++ function), 618  
 sss\_se05x\_mac\_context\_init (C++ function), 618  
 sss\_se05x\_mac\_finish (C++ function), 619  
 sss\_se05x\_mac\_init (C++ function), 619  
 sss\_se05x\_mac\_one\_go (C++ function), 619  
 sss\_se05x\_mac\_t (C++ class), 425  
 sss\_se05x\_mac\_t::algorithm (C++ member), 425  
 sss\_se05x\_mac\_t::cryptoObjectId (C++ member), 425  
 sss\_se05x\_mac\_t::keyObject (C++ member), 425  
 sss\_se05x\_mac\_t::mode (C++ member), 425  
 sss\_se05x\_mac\_t::session (C++ member), 425  
 sss\_se05x\_mac\_update (C++ function), 620  
 sss\_se05x\_mac\_validate\_one\_go (C++ function), 620  
 sss\_se05x\_object\_t (C++ class), 426  
 sss\_se05x\_object\_t::cipherType (C++ member), 426  
 sss\_se05x\_object\_t::curve\_id (C++ member), 426  
 sss\_se05x\_object\_t::isPersistant (C++ member), 426  
 sss\_se05x\_object\_t::keyId (C++ member), 426  
 sss\_se05x\_object\_t::keyStore (C++ member), 426  
 sss\_se05x\_object\_t::objectType (C++ member), 426  
 sss\_se05x\_refresh\_session (C++ function), 621  
 sss\_se05x\_rng\_context\_free (C++ function), 621  
 sss\_se05x\_rng\_context\_init (C++ function), 621  
 sss\_se05x\_rng\_context\_t (C++ class), 426  
 sss\_se05x\_rng\_context\_t::session (C++ member), 426  
 sss\_se05x\_rng\_get\_random (C++ function), 622  
 sss\_se05x\_session\_close (C++ function), 622  
 sss\_se05x\_session\_create (C++ function), 622  
 sss\_se05x\_session\_delete (C++ function), 623  
 sss\_se05x\_session\_open (C++ function), 623  
 sss\_se05x\_session\_prop\_get\_au8 (C++ func-

tion), 624

sss\_se05x\_session\_prop\_get\_u32 (C++ function), 624

sss\_se05x\_session\_t (C++ class), 427

sss\_se05x\_session\_t::ptun\_ctx (C++ member), 427

sss\_se05x\_session\_t::s\_ctx (C++ member), 427

sss\_se05x\_session\_t::subsystem (C++ member), 427

sss\_se05x\_set\_feature (C++ function), 625

sss\_se05x\_symmetric\_context\_free (C++ function), 625

sss\_se05x\_symmetric\_context\_init (C++ function), 625

sss\_se05x\_symmetric\_t (C++ class), 427

sss\_se05x\_symmetric\_t::algorithm (C++ member), 427

sss\_se05x\_symmetric\_t::cache\_data (C++ member), 427

sss\_se05x\_symmetric\_t::cache\_data\_len (C++ member), 427

sss\_se05x\_symmetric\_t::cryptoObjectId (C++ member), 427

sss\_se05x\_symmetric\_t::keyObject (C++ member), 427

sss\_se05x\_symmetric\_t::mode (C++ member), 427

sss\_se05x\_symmetric\_t::session (C++ member), 427

sss\_se05x\_tunnel (C++ function), 626

sss\_se05x\_tunnel\_context\_free (C++ function), 626

sss\_se05x\_tunnel\_context\_init (C++ function), 626

sss\_se05x\_tunnel\_context\_t (C++ class), 428

sss\_se05x\_tunnel\_context\_t::se05x\_session (C++ member), 428

sss\_se05x\_tunnel\_context\_t::tunnelDest (C++ member), 428

sss\_session\_close (C macro), 659

sss\_session\_close (C++ function), 627

sss\_session\_create (C macro), 659

sss\_session\_create (C++ function), 627

sss\_session\_delete (C macro), 659

sss\_session\_delete (C++ function), 628

sss\_session\_open (C macro), 659

sss\_session\_open (C++ function), 628

sss\_session\_prop\_au8\_t (C++ enum), 467

sss\_session\_prop\_get\_au8 (C macro), 659

sss\_session\_prop\_get\_au8 (C++ function), 628

sss\_session\_prop\_get\_u32 (C macro), 660

sss\_session\_prop\_get\_u32 (C++ function), 629

sss\_session\_prop\_u32\_t (C++ enum), 468

sss\_session\_t (C++ class), 428

sss\_session\_t::data (C++ member), 428

sss\_session\_t::extension (C++ member), 428

sss\_session\_t::subsystem (C++ member), 428

SSS\_SESSION\_TYPE\_IS\_SE05X (C macro), 660

sss\_status\_sz (C++ function), 629

sss\_status\_t (C++ enum), 468

SSS\_SUBSYSTEM\_TYPE\_IS\_SE05X (C macro), 660

sss\_symmetric\_context\_free (C macro), 660

sss\_symmetric\_context\_free (C++ function), 630

sss\_symmetric\_context\_init (C macro), 661

sss\_symmetric\_context\_init (C++ function), 630

sss\_symmetric\_t (C++ class), 429

sss\_symmetric\_t::algorithm (C++ member), 429

sss\_symmetric\_t::data (C++ member), 429

sss\_symmetric\_t::extension (C++ member), 429

sss\_symmetric\_t::keyObject (C++ member), 429

sss\_symmetric\_t::mode (C++ member), 429

sss\_symmetric\_t::session (C++ member), 429

SSS\_SYMMETRIC\_TYPE\_IS\_SE05X (C macro), 661

sss\_tunnel (C macro), 661

sss\_tunnel (C++ function), 631

sss\_tunnel\_context\_free (C macro), 661

sss\_tunnel\_context\_free (C++ function), 631

sss\_tunnel\_context\_init (C macro), 661

sss\_tunnel\_context\_init (C++ function), 631

SSS\_TUNNEL\_CONTEXT\_TYPE\_IS\_SE05X (C macro), 662

sss\_tunnel\_dest\_t (C++ enum), 469

sss\_tunnel\_t (C++ class), 429

sss\_tunnel\_t::data (C++ member), 429

sss\_tunnel\_t::extension (C++ member), 429

sss\_tunnel\_t::session (C++ member), 429

sss\_tunnel\_t::tunnelType (C++ member), 429

SSS\_TUNNEL\_TYPE\_IS\_SE05X (C macro), 662

sss\_type\_t (C++ enum), 469

SSSFTR\_SE05X\_AES

command line option, 113

SSSFTR\_SE05X\_AuthEckKey

command line option, 114

SSSFTR\_SE05X\_AuthSession

command line option, 114

SSSFTR\_SE05X\_CREATE\_DELETE\_CRYPTOOBJ

command line option, 114

SSSFTR\_SE05X\_ECC

command line option, 113

SSSFTR\_SE05X\_KEY\_GET

command line option, 114

SSSFTR\_SE05X\_KEY\_SET

- command line option, 114
- SSSFTR\_SE05X\_RSA
  - command line option, 113
- SSSFTR\_SW\_AES
  - command line option, 114
- SSSFTR\_SW\_ECC
  - command line option, 114
- SSSFTR\_SW\_KEY\_GET
  - command line option, 114
- SSSFTR\_SW\_KEY\_SET
  - command line option, 114
- SSSFTR\_SW\_RSA
  - command line option, 114
- SSSFTR\_SW\_TESTCOUNTERPART
  - command line option, 114

## T

- tlvHeader\_t (C++ *class*), 430
- tlvHeader\_t::hdr (C++ *member*), 430

## W

- WithCodeCoverage
  - command line option, 116
- WithExtCustomerCode
  - command line option, 116
- WithNXPNFCRdLib
  - command line option, 116
- WithOPCUA\_open62541
  - command line option, 116
- WithSharedLIB
  - command line option, 116