

1 NXP Plug & Trust Middleware

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

- EdgeLock™ SE050 (**E**, **F** and including *legacy* variants **A**, **B** and **C**)
- EdgeLock™ SE051 (Including variants **A**, **C** and **H**)
- EdgeLock™ SE052

Setting up SE05x development environment for:

- Linux:
 - i.MX: [Section 4.4i.MX Linux Build](#)
 - RaspberryPi: [Section 4.5Raspberry Pi Build](#)
- Microcontroller:
 - [Section 10.6.3Freedom K64F](#)
 - [Section 10.6.4i.MX RT 1060](#)
 - [Section 10.6.5LPC55S69](#)
 - [Section 10.6.6MIMXRT1170](#)
 - [Section 10.6.7FRDM-MCXA153](#)
 - [Section 10.6.8FRDM-MCXN947](#)
- Android: [Section 10.6.9Android: Hikey960](#)
- Windows: [Section 4.1Windows Build](#)

Documentation also covers:

- 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 [Section 5 — Demo and Examples](#).

Dedicated application notes:

- Overview of all available HW, SW and documentation:
 - [SE05x Support Package](#)
- Secure Element configurations are available here:
 - [SE050 Configuration](#)
 - [SE051 Configuration](#)
 - [SE052 Configuration](#)
- Some of the important Application Notes are as shown below:
 - MCU/RTOS: - [K64F - IMXRT1060 - LPC55S69](#)
 - MPU/Linux: - [IMX6UL - IMX8MQ](#)
 - Windows: - [AN12398](#)
- Details regarding SE050 and other detailed application notes can be found at <https://www.nxp.com/products/:SE050>
- Details regarding SE051 and other detailed application notes can be found at <https://www.nxp.com/products/:SE051>
- Details regarding SE052 and other detailed application notes can be found at <https://www.nxp.com/products/:SE052F>



Note:

1. It is recommended to use the latest version of this software available at <https://www.nxp.com/se051>
2. VCOM and PC/SC interfaces (<https://pcscworkgroup.com/>) are provided for quick evaluation of the Secure Element examples and should not be deployed in a product.
3. To contact NXP or to report issues, please use <https://www.nxp.com/support/support:SUPPORTHOME>

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 1060 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 \(and Other MCUs\) Build \(CMake - Advanced\)](#). Steps for i.MX RT 1060 mcxn947 mcxa153 and LPC55S are similar to those for [Freedom K64F \(and Other MCUs\) 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 5.1.1 — DEMO List](#) has an itemized list of the demos.
 - [Section 5.1.1.1 — Platforms List](#)
 - [Section 5.1.1.3 — Cloud connectivity Examples](#)
 - [Section 5.1.1.4 — OpenSSL Engine Examples](#)
 - [Section 5.1.1.5 — mbedTLS Examples](#)
 - [Section 5.1.1.6 — OPC UA Examples](#)
 - [Section 5.1.1.8 — SE05X Specific Examples](#)
 - [Section 5.1.1.10 — Ease of Use Configuration Examples](#)
- Integration with other middlewares/systems is explained in [Section 8 — Plugins / Add-ins](#).
 - [Section 8.1 — Introduction on OpenSSL engine](#)
 - [Section 8.3 — Introduction on MbedTLS ALT \(2.x\) Implementation](#)
 - [Section 10.6.9 — Android: Hikey960](#)
 - [Section 8.7 — Introduction on Open62541 \(OPC UA stack\)](#)
 - [Section 8.9 — PKCS#11 Standalone Library](#)
- The CLI Tool is explained in [Section 9 — CLI Tool](#)
 - [Section 9.7 — 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>
LA_OPT_NXP_Software_License.txt	Contains the End User License Agreement. Required to be agreed mandatory before using the PlugAndTrustMW software
SCR.txt	Software Content Register for external components.

Folder / File Name	Description
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 RT1060.
binaries/PCWindows/ ssscli	This folder contains pre-compiled ssscli tool for Windows platform. Version 03.XX specific DLL is present in binaries/PCWindows/ssscli/03_XX and version 07.02 specific DLL is present in binaries/PCWindows/ssscli/07_02. Copy the required DLL to binaries/PCWindows/ssscli based on the applet version selected.
demos	This folder contains various Demonstration Examples for the supported platforms. See DEMO List for list of supported Demos.
demos / Certificate_ 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
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, SE05x APIs, etc.
hostlib/hostlib/access Manager	This folder contains the source code of the Access Manager application. The Access Manager supports concurrent access from multiple linux processes to an SE05x IoT applet.
nxp_iot_agent	This folder contains source code of the EdgeLock 2GO agent and demonstration examples.
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 (ssscli) for configuring and provisioning the SE05x (Secure Element) Please ensure that PYTHON Version 3.8 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
semslite	This folder contains source code of the SEMS Lite agent, demonstration examples and generator tools.
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 (and Other MCUs) Build (CMake - Advanced)

Platform	Link
RaspberryPi-3 / RaspberryPi-4	Raspberry Pi Build
IMX8M	i.MX Linux Build
RT1060	i.MX RT 1060
RT1170	MIMXRT1170
LPC55S	LPC55S69
MCXN	FRDM-MCXN947
MCXA	FRDM-MCXA153

2 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.

2.2 Known limitations

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

- [SSS] Cipher init manages crypto object: The SSS layer's implementation of multistep symmetric ciphers does not allow concurrent execution of ciphers with the same cipher mode (e.g. twice kAlgorithm_SSS_AES_CBC).
– Workaround : Use lower layer api's instead of sss api's for creating multiple crypto objects.
Refer [se05x Multiple Digest Crypto Objects example](#).

2.3 Release v04.07.01

- New API (`SM_AmResetI2C()`) added in smcom layer (`hostLib/libCommon/infra/sm_api.h`) to send reset request for access manager. Refer *Access Manager command to reset secure element* section in ([Access Manager: Manage access from multiple \(Linux\) processes to an SE05x IoT Applet](#)). - New example to demonstrate sending the reset command from client applications to access manager is added - (`demos/se05x/se05x_am_reset_ex`). Refer - ([SE05X Access Manager Reset example](#))
- Bug fix - smCom socket layer (used for access manager) updated to use connection context. This enables smCom layer to handle sessions from multi threaded applications. - NOTE: It is still mandatory to use access manager when there are multiple threads using secure element. For MCUs, multiple threads need to share the session context. - New example to demonstrate multiple sessions from multi threaded applications is added - (`demos/se05x/se05x_multi_thread_multi_session`). Refer - ([SE05X Multi Thread and Multi session example](#))
- Mbed-TLS build fix for Linux platform when ALT is enabled. (Only static building supported). Dynamic build is disabled for Mbed-TLS as host crypto.
- TLS C Client example scripts updated to support all applet variants now. (CMake file fixed - `demos/linux/tls_client/CMakeLists.txt`). Also the TLS cClient example document ([OpenSSL Engine / Provider: TLS Client example for iMX/Rpi3](#)) updated with TLS client server commands for OpenSSL 3.x / Provider.
- Mbed-TLS configuration guards are used for Mbed-TLS APIs usage in `fsl_sss_mbedtls_apis.c`.
- Bug fix - `se05x_RotatePlatformSCP03Keys` example fails when no SCP03 key file exists.
- Warning message of 'insufficient buffer size' displayed when the log messages are of length greater than 256.
- `seTool` example fixed to handle RSA key id in reference key correctly.

- *func-timeout* library and its usage are removed from ssscli tool. This has no impact on any existing functionalities of ssscli.
- Bug fix - *SSS_FREE* call replaced with *OPENSSL_free* call in *ks_openssl_store_key()* function.
- Bug fix in OpenSSL Provider - Key id parsing from curve parameter string (in *sss_keymgmt_ec_gen_set_params()*) is fixed to handle large key id numbers.

2.4 Release v04.07.00

2.4.1 External module update

- Mbed TLS 3 is updated to version 3.6.2.
- Paho mqtt module in external folder updated to version 1.3.14.
- Protobuf (to auto-generate file for *nxp_iot_agent/src/protobuf/*) version is updated to nanopb-0.3.9.8. (<https://github.com/nanopb/nanopb/blob/nanopb-0.3.9.8/CHANGELOG.txt>).
- **MCU SDK updated to version 2.16**
 - Mbed TLS 2.x is updated to 2.28.5
 - FreeRTOS Kernel updated to version 11.0.1
 - FreeRTOS-plus-TCP updated to version 4.0.0
 - AWS-IoT is updated to use Mbed TLS PSA crypto

2.4.2 New platform support

- MCXN-947 support added (Supports Mbed TLS 3.x version only).
- MCXA-153 support added (No host crypto support available for MCXA-153).

2.4.3 SSSCLI Changes

- **Cryptography module updated to 44.0.0.**
 - Limitation: The following curves are not supported anymore - brainpoolP160r1, brainpoolP192r1, brainpoolP224r1, brainpoolP320r1, secp160k1, secp192k1, secp224k1.
- setuptools module updated to 75.3.0.
- Get memory command to see the available RAM and Flash memory of secure element is added. Refer - [List of ssscli commands](#)

2.4.4 Functional Changes / New feature support

- Platform SCP03 support extended for AES-256 (32 byte) keys. [SE05X Rotate PlatformSCP Keys Demo](#)
- seTool updated for OpenSSL 3.x.y host crypto. (Tested with version - OpenSSL 3.0.14)
- Signal handlers are added to handle signals SIGTERM and SIGINT in access manager and client for clean exit.
- Access manager bug fix - corrected the check for APDU limit error (0x66A6).
- Pre-built vcom and get info binaries added for MCXA-153 and MCX-947 boards
- T=1ol2C Stack updated to remove the endianness dependent code.
- smComSocket_Transceive functions are updated to retry on EINTR error also.
- **Fixes for memory leaks.**
 - emsa_decode_and_compare function
 - PKCS11 module
 - Access Manager exit

- Fixes for static analysis findings.
- OpenSSL Provider updated to version 1.1.2 (Provider code is also uploaded at - <https://github.com/NXPPlugNTrust/se05xOpenssl-provider>)

- **1.1.2 updates**

- ECDSA functions handle different formats of SHA algorithm string (Example - "sha256" / "SHA256" / "SHA2-256")

- **1.1.1 updates**

- Bug fix: Signature length calculation fixed in ECDSA functions.

- **1.1.0 updates**

- CSR generation extended for all SHA algorithms now. (Get context functions - 'sss_rsa_signature_get_ctx_params' and 'sss_ecdsa_signature_get_ctx_params' updated to handle all SHA algorithms).
 - TLS use case : Ephemeral key is generated on host by default. (Enable 'SSS_ENABLE_SE05X_EC_KEY_GEN_WITH_NO_KEYID' in sssProvider_key_mgmt_ec.c file to generate on secure element).
 - Performance improvement : Provider is updated to store the client / server public key on host to avoid multiple secure element reads during TLS connection.
 - Bug fix - Correct RSA encrypt algorithm is set in function sss_rsa_enc_set_ctx_params, when 'OSSL_ASYM_CIPHER_PARAM_PAD_MODE' parameter data is integer type.
 - OSSL algorithms are updated with algorithm properties.
 - Enable / Disable random number generation in nxp provider using compile time option - 'SSS_PROV_DISABLE_SE05X_RNG'. (Disabled by default)
 - Key id for RSA key generation can be passed to provider via 'OSSL_PKEY_PARAM_RSA_FACTOR2' parameter.
 - ECC / RSA key management import functions added - to handle reference keys. If the input key is not reference key, the function returns error to roll back on other available providers.
 - ECC key management - match and duplicate functions added.
 - ECDSA digest verify support added (function - sss_ecdsa_signature_digest_verify).
 - RSA digest verify support added (function - sss_rsa_signature_digest_verify).
 - RSA generate Key feature will generate RSA-CRT type keys by default. (Use compile time define SSS_PROV_GENERATE_RSA_PLAIN to change the RSA key type to plain.)

- **1.0.3**

- ECC and RSA TLS use case extended for reference keys in file format also.

- Linux TLS Client Server examples, AWS and Azure cloud examples are updated to work with OpenSSL Provider also (in addition to existing OpenSSL engine support)
- PKCS11 updated to version to 4.8 (PKCS11 code is also uploaded at - <https://github.com/NXPPlugNTrust/se05x-pkcs11>)

- **4.8 Updates**

- Global TLS context initialization is removed as cloud example doesn't have dependency on TLS context
 - PKCS11 session handling clean up to support only single session
 - HKDF support added for PKCS11 specification version greater than or equal to 3.x
 - RSA_CRT support added for RSA key generation, sign/verify. (compile with PKCS11_ENABLE_RSA_KEY_GEN_CRT cmake option for CRT. By default plain RSA is enabled)

- **4.7 Updates**

- Breaking: KeyID Handling: CKA_ID attribute (-id) passed is treated as byte array to avoid byte swapping. ID can be passed without changing the endianness now.
 - EC curve support extended for NIST-K and Brainpool.
 - PBKDF2 support added.

- Bug fix: ASN-1 conversion in C_Verify function. PKCS11 implementation can now handle both raw and ASN-1 signature format for ECDSA verify.

2.4.5 SEMS Lite Change

- Comparison of public key from secure element and SEMS Lite script (PkSemsCaAut field) is done to determine whether the applet can be updated.

2.4.6 File/Folder relocation

- se05x_apis.h is renamed to se05x_reset_apis.h (file path - simw-top/hostlib/hostLib/platform/inc).

2.4.7 Build system changes

- New cmake option *PTMW_SE_RESET_LOGIC* is added to select the reset logic for SE051/SE052.
- *frdmmcxn947* and *frdmmcxn153* options are added for *PTMW_Host* cmake option.
- New cmake options for EdgeLock 2GO hostname and port selection (*EDGELOCK2GO_HOSTNAME* and *EDGELOCK2GO_PORT*)

2.4.8 APIs & enum/types Changes

- se05x_ic_reset API updated to receive applet version as input parameter. This is used to decide the reset logic for SE051 and SE052. (file path - simw-top/hostlib/hostLib/platform/inc/se05x_reset_apis.h)

2.4.9 EdgeLock 2GO Agent Changes

- EdgeLock 2GO Agent enablement on top of OpenSSL 3.x using built-in sssProvider
- EdgeLock 2GO hostname and port selection via command line or environment variable (only Windows/Linux targets) or preprocessor define.

2.5 Release v04.06.00

• Mbed TLS 3.5.0 support added in middleware

- New cmake option added to choose between Mbed TLS 2.x and Mbed TLS 3.x. (*PTMW_MBedTLS:STRING=2_X* / *PTMW_MBedTLS:STRING=3_X*).
- **Mbed TLS 3.x ALT files added. (Using reference keys to use secure element). Refer [Introduction on Mbed TLS \(3.x\) ALT Implementation](#). Crypto operation supported -**
 - ECDSA Sign and Verify
 - RSA Sign and Verify
 - Random number generation
 - ECDH
- New examples : Mbed TLS 3.x Client / Server. Refer section - ‘Testing Mbed TLS ALT’ in [Introduction on Mbed TLS \(3.x\) ALT Implementation](#)

2.6 Release v04.05.03

- Fixes for static analysis findings
- **OpenSSL Provider (v1.0.2) updates**

- TLS support extended for RSA keys. (Refer [OpenSSL Engine / Provider: TLS Client example for iMX/Rpi3](#))

- RSA Encrypt and decrypt feature added in provider. (Refer [Introduction on OpenSSL_provider](#)). Also refer the example script - `simw-top/sss/plugin/openssl_provider/scripts/openssl_RsaEnc.py`

• PKCS11 (v4.6) updates

– Bug fixes for

- Digest of large input data (>1024) is handled using multistep (Refer Digest functions in `simw-top/sss/plugin/pkcs11/sss_pkcs11_pal_core.c`).
- HMAC of large input data (>1024) is handled using multistep (Refer C_Sign and C_Verify functions in `simw-top/sss/plugin/pkcs11/sss_pkcs11_pal_core.c`).
- List object for uses cases where secure element has more than 200 objects.
- Removed intermediate buffers used in RSA Encrypt / decrypt feature.
- ECDSA multi-step operation support added.
- RSA Sign and verify multi-step operation support added.
- Test scripts updated with OpenSSL counterpart testing. (Refer `simw-top/sss/plugin/pkcs11/scripts`).
- Fixes for memory leaks and static analysis findings.

2.7 Release v04.05.02

- New FiRaLite API `se_FiRaLite_API_InitiateTransaction_extended()` added (accept P1 as input parameter).
- Input argument `void *conn_ctx` changed to '`pSe05xSession_t session_ctx`' in APIs of `se_FiRaLite_API.h` and `sus_APDU.h`.
- Fixes for static analysis findings

2.8 Release v04.05.01

- SE052 secure element support added. Refer [SE052 specific notes](#).
- Deep power down (T1ol2C command) API added for SE052 - `phNxpEse_deepPwrDown()`.
- New examples added - [SE052 Reset Example](#) and [SE052 Deep Power down example](#)
- I2C Wrapper for Linux/RT1060/RT1170 : I2C read back-of-delay extended to additional platforms including Linux. This leads to consistent timeout of ~20 seconds across all platforms when there is no response from secure element.
- T=1ol2C : During session open, added logic in T=1ol2C to wait till previous transaction is complete (by sending WTX response to up to 40 WTX requests). This helps in case the previous transaction was not completed or middleware was re-started.
- SEMS Lite : Signature over command mismatch is ignored with warning message. The change is required to handle the tearing events of SEMS Lite scripts.
- I2C Wrapper for RT1060/RT1170 : Add workaround to send I2C Master stop if NAK received in I2C driver (`simw-top/hostlib/hostLib/platform/ksdk/i2c_imxrt.c`)
- NVM write warning log messages are added in SE05x APDU APIs. (in release v04.05.00).
- SE05X EdgeLock 2GO Agent
 - Debug logging reverted to disabled by default.
 - Fix in Remote Provisioning Server object id array conversion

2.9 Release v04.05.00

2.9.1 Build system changes

- A71CH related cmake options are removed. (PTMW_A71CH_AUTH, PTMW_Applet:STRING=A71CH, PTMW_SCP:STRING=SCP03_HostCrypto)

- Applet version *06_00* removed from *PTMW_SE05X_Ver*

2.9.2 Functional Changes

- A71CH support removed.
- Applet version *06_00* support removed.
- Private key (ECC and RSA) create disabled at SE05x APDU layer
- BN Curve and ECDA API removed.
- Thread safety when using SCP03 in SSS layer. (Disabled by default. To enable, uncomment `//#define SSS_USE_SCOP3_THREAD_SAFETY` in `fsl_sss_se05x_apis.c`)
- Access manager update - Applet reselect on 0x6D00 error
- Access manager session reopen on APDU throughput (0XA66) error
- ENGINE_load_private_key is supported in openssl engine
- Issue of policies passed to the SSS API being modified - is Fixed.
- Fixes for compilation warnings and static analysis findings.
- Mbedtls in external components updated to version 2.28.3
- PKCS11 - Added support for multi step sign/verify with rsa and ecc key.
- PKCS11 - Added support for HMAC-SHA256 operations.
- PKCS11 - Key label handling - The endianness of the key id passed in the key label (-label) is not changed. Example for sss:20181001, keyID used is 20181001.

2.9.3 Examples / DEMO updates

- TLS C example extended for openssl 3.0 provider
- SE Tool example updated to support RSA reference keys. See: [Section 5.8.1 Tool to create Reference key file](#).
- GCP and Ibm_watson cloud examples deprecated

2.9.4 APIs & enum/types Changes

- Input pointers are updated with const qualifiers for asymmetric_sign APIs
`(sss_se05x_asymmetric_sign_digest, sss_se05x_asymmetric_sign,`
`sss_se05x_asymmetric_verify_digest, sss_se05x_asymmetric_verify,`
`sss_asymmetric_sign_digest, sss_asymmetric_verify_digest)`

2.10 Release v04.04.02

2.10.1 Changes

- Bug fix : applet_version not preserved in sss_session_auth_open
- Additional error checking in sss_se05x_TXn()
- Bug fix : failing phNxpEse_chipReset does not return failure
- phNxpEse_open memory leak fix
- APDU throughput error code (SM_ERR_APDU_THROUGHPUT) added in smStatus_t.
- APDU throughput error code (kStatus_SSS_ApduThroughputError) added in sss_status_t. SSS APIs are updated return new throughput error code.
- Reset example added.

2.11 Release v04.04.01

2.11.1 Changes

- New APDUs for RSA sign and verify with salt length added - [Se05x_API_RSASign_WithSalt\(\)](#) and [Se05x_API_RSAVerify_WithSalt\(\)](#).
- Allow data communication in `sss_se05x_TXn` function even when applet selection is skipped.
- Fixes for compilation warnings and static analysis findings.

2.12 Release v04.03.03

2.12.1 Changes

- T=1oI2c fix - Added retry code for `I2C_FAILED` error code also in Linux like systems
- Memory leak fixes in openssl and mbedtls SSS hostcrypto layer
- openssl 3.0 provider updates for EC TLS use case

2.13 Release v04.03.02

2.13.1 Functional Updates

- Added parameter `uint32_t applet_version` in structure `Se05xSession_t`
 - Parameter `applet_version` is updated on successful selection of applet in API [sss_se05x_session_open\(\)](#)
- Updated SE05x API [Se05x_API_GetFreeMemory\(\)](#)
 - Argument `pfreeMem` is now `uint32_t` (*earlier it was uint16_t*)
 - All demos that refer to [Se05x_API_GetFreeMemory\(\)](#) are updated with this datatype change
- Updated SE05x API [Se05x_API_HKDF_Extended\(\)](#)
 - External salt is mandatory. This is checked using new parameter `applet_version`. If `applet_version` is newer than `0x07020000`, then we allow setting salt even if HKDF mode is `ExpandOnly`
- Demo update for [Section 5.7.2 SE05X Get Info example](#).
 - With new JCOP, one extra TLV is returned in `JCOPIdentify` command
 - Added parsing of new TLV : `JCOP OS Core ID`
 - Parsing is done based on new parameter `applet_version`. If `applet_version` is newer than `0x07020000`, we parse the extra TLV, otherwise we continue parsing remaining data

2.14 Release v04.03.01

2.14.1 Important Security Updates

- Added security fixes on 24-Feb-2023 to prevent buffer overflow on the T=1oI2C stack. It is important to use the updated “hostlib/hostLib/libCommon/smCom/T1oI2C/” and smCom module.

2.14.2 Other Miscellaneous Changes

- PKCS#11 : Fixed SECP521R1 key parsing bug

2.15 Release v04 . 03 . 00

2.15.1 Build system changes

- Added OpenSSL version selection (1.1 and 3.0) in cmake build system

2.15.2 APIs & enum/types Changes

- API `Se05x_API_ECDAAASign()` marked as deprecated. This will be removed in next release.
- Cipher type `kSSS_CipherType_EC_BARRETO_NAEHRIG` marked as deprecated. This will be removed in next release.
- enum `SE05x_ECDAAASignatureAlgo_t` marked as deprecated. This will be removed in next release.
- [`Se05x_API_WriteECKey\(\)`](#) and [`Se05x_API_WriteRSAKey\(\)`](#) with P1_PRIVATE is now deprecated. This will be removed in next release.
- T4T APDUs added in hostlib

2.15.3 Functional Changes

- [`sss_se05x_asymmetric_sign_digest\(\)`](#) implementation changed to use digest from host crypto for RSA sign.
- pkcs11 sign and verify operations will use host crypto for digest calculation.
- Bug fix : pkcs11 will now list key pair as 2 objects (private and public).

2.15.4 New feature support

- OpenSSL 3.0 support. OpenSSL provider implementation provided to access secure element (SE05x).
- OpenSSL 3.0 support in SSS OpenSSL layer.

2.15.5 SSSCLI Changes

- Bug fix : read id list fix

2.15.6 Other Miscellaneous Changes

- Added a check to prevent a potential buffer overflow issue in T=1I2C stack

2.16 Release v04 . 02 . 00

2.16.1 Build system changes

- Added support for powershell (`scripts/env_setup.ps1`) to set environment variables for generating build files. Refer - [Windows Build](#)
- Default applet in build configuration changed to SE050E.

2.16.2 APIs & enum/types Changes

- Extended `smStatus_t` with new error codes
- Updated behaviour of `sss_se05x_key_object_get_handle()` to return a success and print warning if it is unable to read attributes but the object exists so that other operations (like deleting) can proceed if they don't depend on object attributes.

2.16.3 Functional Changes

- Updated OEF specific SCP keys handling. See [Section 3.9.1.1 Configuring for OEF specific Platform SCP keys](#)
- mbed-crypto removed and PSA support moved to mbedTLS v2.26.0. See [Section 8.5 Platform Security Architecture](#)

2.16.4 New feature support

- SE050E applet support added. (Enabled by default in build configurations).
- SE051-H applet support added (Provides PAKE support).
- OpenSSL engine can be build with 'host crypto = mbedtls' and 'host crypto = user' also.

2.16.5 SSSCLI Changes

- Added precompiled libraries for Applet version 03.XX and 07.02 in `binaries/PCWindows/ssscli/03_XX` and `binaries/PCWindows/ssscli/07_02`. Default configuration is set to 03.XX

2.16.6 Examples / DEMO updates

- Added Secure Authenticator (Qi) demo [Section 5.7.31 Secure Authenticator \(Qi\) Authentication demo](#)
- Default port for access manager clients changed to 8040.
- Added read and delete policies to authentication objects created by [Section 5.7.27 Delete and Test Provision](#)

2.16.7 Other Miscellaneous Changes

- Access manager (with Unix sockets) clients to stop when access manager process is killed.
- FD_CLOEXEC option set when access manager opens unix socket. (*FD_CLOEXEC* flag specifies that the file descriptor should be closed when an exec function is invoked).
- Bug fix : Memory leak fix on open session with wrong keys
- Bug fix : Linux VCOM driver communicates very slow.
- Session handling cleaned up for PKCS11 library. Session ID is returned to the application instead of session handle pointer. Added mapping table from session handle to session handle pointer and added sanity checks before using the session for any operation. See [Section 8.9 PKCS#11 Standalone Library](#)

2.17 Release v04.01.03

2.17.1 Other Miscellaneous Changes

- Added documentation to configure MW for SE050E variant. `SE050E-documentation`

2.18 Release v04.01.01

2.18.1 SEMSLite

- Semslite header updated from a manually edited JSON file.

2.18.2 Other Miscellaneous Changes

- Fixed Prebuilt binaries for ssscli

2.19 Release v04.01.00

2.19.1 SEMSLite

- Semslite example updated for OEF A8FA.

2.19.2 Examples / DEMO updates

- EDDSA example added. See: [Section 5.2.1.10EDDSA Example](#).
- SCP03 session resume example added. See: [Section 5.7.30SE05X SCP03 BOOT Example](#).

2.19.3 Other Miscellaneous Changes

- User crypto implementation extended for AES ECB encrypt/decrypt
- Added PlatformSCP keys of the generic types
- PKCS#11: Refactored library with bugfixes
- PKCS#11: Added support to cache object entries during `C_FindObjects()` to improve execution time. Max cache entries controlled by macro `USER_MAX_ID_LIST_SIZE`.
- Bugfix: Access manager clients to stop when access manager process is killed.

2.20 Release v04.00.01

2.20.1 Build system changes

- New cmake option to enable unix sockets in access manager - `-DWithAccessMgr_UinxSocket:BOOL=`

2.20.2 Functional Changes

- Added unix socket support for access manager

2.21 Release v04.00.00

2.21.1 Build system changes

- Support for new applet version 7.x

2.21.2 APIs & enum/types Changes

- Policy changes for 7.x applet (Also refer - [Policies](#))
 - Below policies removed from `sss_policy_sym_key_u` for applet version 7.x.
 - Allow key derivation policy (`can_KD`)
 - Allow to write the object policy (`can_Write`)
 - Allow to (re)generate policy (`can_Gen`)
 - Below policies are added for `sss_policy_sym_key_u` for applet version 7.x.
 - Allow TLS PRF key derivation (`can_TLS_KDF`)
 - Allow TLS PMS key derivation (`can_TLS_PMS_KD`)
 - Allow HKDF (`can_HKDF`)
 - Allow PBKDF (`can_PBKDF`)
 - Allow Desfire key derivation (`can_Desfire_KD`)
 - Forbid External iv (`forbid_external_iv`)

- Allow usage as hmac pepper (`can_usage_hmac_pepper`)
- Below policies removed from `sss_policy_asym_key_u` for applet version 7.x.
 - Allow to read the object policy (`can_Read`)
 - Allow to write the object policy (`can_Write`)
 - Allow key derivation policy (`can_KD`)
 - Allow key wrapping policy (`can_Wrap`)
- Below policies are added for `sss_policy_common_u` for applet version 7.x.
 - Allow to read the object policy (`can_Read`)
 - Allow to write the object policy (`can_Write`)
- Added new policy - `ALLOW_DESFIRE_CHANGEKEY`, `sss_policy_desfire_changekey_authId_value_u`
- Added new policy - `ALLOW_DERIVED_INPUT`, `sss_policy_key_drv_master_keyid_value_u`
- **can_Read** and **can_Write** polices are moved from symmetric and asymmetric object policy to common policy in applet 7.x. **PLEASE UPDATE THE APPLICATIONS ACCORDINGLY.**
- New attestation scheme for applet 7.x
 - Updated API `Se05x_API_TriggerSelfTest_W_Attst()` for applet version 7.x.
 - Updated API `Se05x_i2c_master_attst_txn()` for applet version 7.x.
 - Updated API `sss_se05x_key_store_get_key_attst()` for applet version 7.x.
- New API added for PBKDF2 support: `Se05x_API_PBKDF2_extended()`. Supports optional salt object id and optional derived object id.
- New mode `kMode_SSS_Mac_Validate` added to support MAC validation feature in `sss_mac_one_go()` and `sss_mac_* multistep` APIs.
- New API added for ECDH calculation with option to select ECDH algorithm: `Se05x_API_ECDHGenerateSharedSecret_InObject_extended()`. ECDH algorithms supported - `EC_SVDP_DH` and `EC_SVDP_PLAIN`.
- New API added `sss_cipher_one_go_v2()` with different parameters for source and destination lengths to support ISO/IEC 9797-M2 padding.
- Internal IV generation supported for AES CTR, AES CCM, AES GCM modes:
`kAlgorithm_SSS_AES_GCM_INT_IV`, `kAlgorithm_SSS_AES_CTR_INT_IV`,
`kAlgorithm_SSS_AES_CCM_INT_IV`.
- New MAC algorithm - `kAlgorithm_SSS_DES_CMAC8` supported.
- New api `Se05x_API_ECPPointMultiply_InputObj()` added.
- New api `Se05x_API_WriteSymmKey_Ver_extended()` added to set key with minimum tag length for AEAD operations
- Removed all deprecated defines starting with `With` and replaced with `SSS_HAVE_`

2.21.3 Functional Changes

- ECKey authentication is updated to read SE.ECKA public key with attestation using `Se05x_API_ReadObject_W_Attst_V2()` or `Se05x_API_ReadObject_W_Attst()` (based on applet version) instead of GetData APDU. To authenticate the public key read with attestation, signature verification is performed on the data received from SE. See details of `Se05x_API_ReadObject_W_Attst_V2()` / `Se05x_API_ReadObject_W_Attst()`.

2.21.4 New platform support

- [Section 10.6.6MIMXRT1170](#) platform support added.

2.21.5 SSSCLI Changes

- Python version 3.9 supported
- Applet 7.x version policies updated

2.21.6 Examples / DEMO updates

- Attestation examples updated to handle new attestation scheme of applet 7.x. See: [Section 5.2.1.8ECC NIST256 Key Attestation Example](#), [Section 5.2.1.9ECC MONTGOMERY-25519 Key Attestation Example](#), [Section 5.7.13Read object with Attestation](#), [Section 5.7.7I2C Master Example](#)
- Example added to demonstrate FIDO protocol. se05x-fido-ecdaa [se05x-fido-ecdaa](#)
- Example added [Section 5.7.29SE05X Allow Without SCP example](#)
- Rotate PlatformSCP03 keys (se05x_RotatePlatformSCP03Keys) example with openssl host crypto - Fixed.

2.21.7 External modules Changes

- MCU-SDK updated to SDK version 2.10.0
- mbedTLS updated to version 2.26.0
- Amazon-FreeRTOS updated to version 202012.00
- OpenSSL windows precompiled binaries updated to 1.1.1l

2.21.8 Other Miscellaneous Changes

- sss_se05x_cipher_update() and sss_se05x_aead_update() APIs modified to use input buffer directly.
- Bugfix: Write of large binary files with policy fails on applet 3.x.

2.22 Release v03.03.00

2.22.1 Build system changes

- Option added to allow create-cmake-project.py to run deterministically for Raspberry Pi [Raspberry Pi Build](#)

2.22.2 Functional Changes

- Added call to xLoggingTaskInitialize() to enable FreeRTOS prints.

2.22.3 New feature support

- Extended access manager to support A71CH. Refer Build options for A71CH in Access Manager in [Access Manager: Manage access from multiple \(Linux\) processes to an SE05x IoT Applet](#)

2.22.4 SSSCLI Changes

- Added API to set and get binary data. Refer - [List of ssscli commands](#)

2.22.5 Examples / DEMO updates

- Added example to demonstrate managing multiple crypto objects from application - se05x_MultipleDigestCryptoObj
- se05x_ConcurrentEcc demo modified to use runtime generated key

2.22.6 Other Miscellaneous Changes

- `sss_openssl_cipher_one_go()` api modified to use EVP calls for AES (ECB, CBC, CTR)
- `sss_se05x_cipher_update()` api modified to use block size of 256 to enhance performance.
- Session open retry feature added for accessManager clients - (Only with `-DSMCOM:STRING=JRCP_V1_AM` build option)

2.23 Release v03.01.01

2.23.1 File/Folder relocation

- N.A.

2.23.2 Build system changes

- N.A.

2.23.3 APIs & enum/types Changes

- N.A.

2.23.4 Functional Changes

- “READY=1” start up notification by access manager is disabled by default. Refer - [Access Manager: Manage access from multiple \(Linux\) processes to an SE05x IoT Applet](#)

2.23.5 New platform support

- N.A.

2.23.6 New feature support

- N.A.

2.23.7 SEMSLite

- N.A.

2.23.8 SSSCLI Changes

- Prebuilt ssscli binary updated to support both SE050 and SE051

2.23.9 Documentation Changes

- N.A.

2.23.10 Examples / DEMO updates

- N.A.

2.23.11 Communication Layer Changes

- N.A.

2.23.12 EdgeLock 2GO agent

- N.A.

2.23.13 User Interface Changes

- N.A.

2.23.14 External modules Changes

- N.A.

2.23.15 Other Miscellaneous Changes

- N.A.

2.24 Release v03.01.00

2.24.1 File/Folder relocation

- N.A.

2.24.2 Build system changes

- Updated build tools installation for rpi build.

2.24.3 APIs & enum/types Changes

- Extended kSSS_KeyPart_Default for other objectType.
 - Earlier: Object type kSSS_KeyPart_Default is used for Binary Files, Certificates, Symmetric Keys, PCR and HMAC-key.
 - Now: UserID and Counter are added for kSSS_KeyPart_Default. This means objectType of UserID and Counter will be kSSS_KeyPart_Default after calling [sss_key_object_get_handle](#). Comment for enum sss_key_part_t is updated accordingly.
- Added new API [Se05x_API_WritePCRWithType\(\)](#) with support to write transient PCR objects also.
- Deprecated API [Se05x_API_WritePCR\(\)](#). Added macro [ENABLE_DEPRECATED_API_WritePCR](#) to enable compilation of deprecated API [Se05x_API_WritePCR\(\)](#). Support will be removed by Q1 2022.
- Bugfix - Handling of result tag in case of failure in [Se05x_API_AeadOneShot\(\)](#), [Se05x_API_AeadFinal\(\)](#) and [Se05x_API_AeadCCMFinal\(\)](#)

2.24.4 Functional Changes

- Cleanup for heap management macros. Added support to redirect macros to FreeRTOS APIs. [Section 3.18SSS Heap Management](#).
- Changed from Heap_3 to Heap_4 for [Section 10.6.3Freedom K64F](#) and [Section 10.6.4i.MX RT 1060](#)
- Bugfix - KVN12 key can be used for PlatformSCP authentication now in SE051.
- Access manager will send “READY=1” start up notification to systemd. (Enabled by default on iMX and Rpi platforms)
- AKM - Improved error handling
- SE05x APDU - Response length set to 0 in error condition - [t1vGet_u8buf\(\)](#).
- Created separate library (mwlog) for logging framework. See [Section 3.7Logging](#)
- Order of log level reversed. Current log level is - { "ERROR", "WARN ", "INFO ", "DEBUG" }.

- Added reserved commands for access manager - hostlib/hostLib/accessManager/inc/accessManager.h.

```
// For future use

#define RESERVED_ID2          0x61
#define RESERVED_ID3          0x62
#define RESERVED_ID4          0x63

#define RESERVED_ID5          0x70
#define RESERVED_ID6          0x71
#define RESERVED_ID7          0x72
#define RESERVED_ID8          0x73
```

2.24.5 New platform support

- N.A.

2.24.6 New feature support

- Mbedtls ALT is extended with ECDSA verify operation using MBEDTLS_ECDSSA_VERIFY_ALT define. (Disabled by default). Using this all EC public key verify operations can be performed using SE05x.

2.24.7 SEMSLite

- N.A.

2.24.8 SSSCLI Changes

- N.A.

2.24.9 Documentation Changes

- N.A.

2.24.10 Examples / DEMO updates

- Added Secure Boot demo for LPC55S. Refer [Section 3.19Secure Boot](#).

2.24.11 Communication Layer Changes

- New smcom option -DSMCOM:STRING=JRCP_V1_AM added for client (using access manager) build. When using multiple clients with user authentication, this will ensure the user session is established in an atomic way per client connect.

2.24.12 EdgeLock 2GO agent

- Updated RTP server demo example to support RSA keys and Static Public Key for offline remote trust provisioning.

2.24.13 User Interface Changes

- N.A.

2.24.14 External modules Changes

- N.A.

2.24.15 Other Miscellaneous Changes

- Changed files under BSD3 License with NXP Copyright to Apache2 License.
- Changed files under Proprietary license to Apache 2 License.

2.25 Release v03.00.06

2.25.1 Examples / DEMO updates

- GetInfo to use Channel 0 instead of Channel 1 to query information from JCOP.
- Azure demo in KSDK using FreeRTOS needs more heap to perform data exchange. Heap increased to fix the issue.

2.25.2 EdgeLock 2GO agent

- EdgeLock 2GO agent demo example now provides report of provisioning result.
- Added client - server examples to demonstrate offline remote trust provisioning.
- Added support for i.MX RT1060.

2.26 Release v03.00.05

2.26.1 Functional Changes

- Added Platform SCP03 keys for SE051 (Variant A2 and C2).

2.26.2 New feature support

- Added support for Key Attestation when multiple digests are specified during key creation in Android Keystore.

2.26.3 SSSCLI Changes

- Read ID list has been sorted based on object index.

2.26.4 Documentation Changes

- Updated documentation to include links to SE051 specific documents and Application Notes.

2.26.5 Examples / DEMO updates

- Added examples [Section 5.7.25ECC Concurrent Example](#) and [Section 5.7.26Symmetric Multi Step Concurrent Example](#) for Access Manager
- Access Manager ([Access Manager: Manage access from multiple \(Linux\) processes to an SE05x IoT Applet](#)): miscellaneous implementation improvements.

2.27 Release v03.00.04

2.27.1 Build system changes

- Building the OpenSSL Engine with `OPENSSL_NO_HW` defined is now triggering a build error. This changes the old behaviour where the OpenSSL Engine built did not support integration with the Secure Element.

2.27.2 SSSCLI / PyCLI Changes

- Added support for session authentication with Platform SCP03 combinations:
 - `UserID_PlatformSCP03`
 - `AESKey_PlatformSCP03`
 - `ECKey_PlatformSCP03`

2.27.3 Documentation Changes

- SEMS Lite documentation update
- Access Manager documentation update
- Yocto flow to create iMX8 SD card updated to Yocto Zeus

2.27.4 Examples / DEMO updates

- `demos/se05x/se05x_ReadWithAttestation`: Updated parsing of Object attributes. Take into account differences between SE050 and SE051 regarding Object attributes. ([Read object with Attestation](#))
- `hostlib/hostlib/accessManager`: Additional functionality and miscellaneous bug improvements on an embedded Linux system. ([Access Manager: Manage access from multiple \(Linux\) processes to an SE05x IoT Applet](#))
 - Added command line option to allow incoming connections on all IP addresses, previously this was the default behaviour
 - Improvement: Server now closes STREAM connection in case Client has closed it.
 - Improvement: Fix applet version detection
 - Compile time removal (default) of incomplete and un-tested lock / unlock handling

2.27.5 Other Miscellaneous Changes

- Bug Fix : Fix for attestation read of symmetric objects which have no read policy.

2.28 Release v03.00.03

2.28.1 Build system changes

- Embedded Linux: Support for building against mbedTLS shared libraries

2.28.2 APIs & enum/types Changes

- `smCom_Init`: return type is now `U16` instead of `void`. Return value indicates success/failure to create mutex/semaphore.
- The enumerated type `SE05x_EDSignatureAlgo_t` contained a value `kSE05x_EDSignatureAlgo_ED25519_PH_SHA_512`. The mnemonic name of the value was misleading as it actually corresponded to the *Pure*

EDDSA algorithm not the *Prehashed (PH) EDDSA algorithm*. This has now been corrected. **This will require corresponding update in the application code.**

- EDDSA signature algorithm enumerated value `kSE05x_EDSignatureAlgo_ED25519PH_SHA_512` is changed into `kSE05x_EDSignatureAlgo_ED25519PURE_SHA_512`.
- EDDSA attestation algorithm enumerated value `kSE05x_AttestationAlgo_ED25519PH_SHA_512` is changed into as `kSE05x_AttestationAlgo_ED25519PURE_SHA_512`.

2.28.3 New platform support

- Embedded Linux / Yocto: using Yocto Zeus (5.4.24_2.1.0) as iMX reference environment
- Integrated KSDK Rel.12 specific changes into Plug&Trust MW

2.28.4 New feature support

- Multithreaded support extended from Posix to FreeRTOS
- SE05x Garbage collection API and example using the Garbage collection API

2.28.5 SEMSLite

- Support for MulticastPackageFormat 1.2
- KSDK Examples added
- SEMSLite CLI application now available as a pre-compiled Windows binary

2.28.6 Examples / DEMO updates

- `hostlib\hostlib\accessManager` illustrates how multiple processes can share access to a Secure Element on an embedded Linux system. ([Access Manager: Manage access from multiple \(Linux\) processes to an SE05x IoT Applet](#))

2.28.7 Communication Layer Changes

- Select Applet: No prequel with Select CM required for SE05x

2.28.8 External modules Changes

- `simw-top/ext/amazon-freertos` updated to KSDK version 202002.00_rev1
- `simw-top/ext/mbedtls` updated to KSDK version 2.16.2_rev2
- `simw-top/ext/mcu-sdk` updated to public release 2.8.0
- `simw-top/ext/paho.mqtt.c` updated to version 1.3.5

2.28.9 Other Miscellaneous Changes

- Fixed warnings when compiling with GCC 9.2
- Fixed typo in example code API: `ex_sss_keystore_and_object_init()` is now `ex_sss_key_store_and_object_init()`
- Extended SE051 specific APDU command and response buffer size to match SE051's capabilities.
- Removed support for intermediate versions of the SE051 product
- SSS API blocks SHA512 attestation, signing and verification for RSA512 key

2.29 Release v03.00.02

2.29.1 Build system changes

- Cmake workareas for iMX/RspPi created by scripts/create_cmake_projects.py now default to linking against shared libraries
- Shared libraries installed by Cmake all go into /usr/local/lib (paho libraries no longer go into /usr/local/lib64)
- New Host Crypto library cmake option MBEDCRYPTO added: mandatory for PSA TF-M demo ([Platform Security Architecture](#)). UseMBEDTLS for other use cases and demos where mbedTLS host crypto is required.
- Supported Secure element versions (SE05X_Ver) are 03_XX (SE050) and 06_00 (SE051) only.

2.29.2 APIs & enum/types Changes

- AEAD APIs are added as a part of Se051.

[sss_aead_context_free\(\)](#)
[sss_aead_context_init\(\)](#)
[sss_aead_finish\(\)](#)
[sss_aead_init\(\)](#)
[sss_aead_one_go\(\)](#)
[sss_aead_update\(\)](#)
[sss_aead_update_aad\(\)](#)

2.29.3 Functional Changes

- Added heap management macros for SSS layer. [SSS Heap Management](#)
- Extend support to SE051 (default build is for SE050, to make SE051 specific features available in the host software, set SE05X_Ver to 06_00)

2.29.4 New feature support

SEMS Lite - Enables the deployment and update of applets in the field, while enabling preservation of on device applet data.

2.29.5 SSSCLI / PyCLI Changes

- ssscli connect se050 has been replaced with ssscli connect se05x

2.29.6 Examples / DEMO updates

- Examples Added for Se051
 - se05x_ReadState: [SE05X Read State example](#)
 - se05x_Perso_Delete_Mod_RSAKeyGen: [SE05X Personalization Remove RSA Key Generation Module](#)
- iMX/RspPi Linux cloud demos now link against Paho libraries that are included in the Plug&Trust MW in the siwm/ext folder. Except for Azure cloud demo which links against downloaded Paho from git master.
- PSA TF-M demo on LPC55S69 ([PSA Non Secure Example](#) or [Platform Security Architecture](#))

2.29.7 Other Miscellaneous Changes

- T1oI2C:
 - Fixed: potential null pointer dereference
 - Fixed: RSYNC _ + CRC error results in saving response to uninitialised buffer.
- Yocto recipe for Plug&Trust Mw Package: contained in scripts/yocto folder

- hostlib/hostLib/platform/linux/i2c_a7.c: A call to `axI2CTerm()` now closes the I2C file descriptor associated with the I2C communication channel.
- Fix OpenSSL Engine Issue: Key Creation was not handed back to SW implementation (issue introduced in v02.16.00 and linked to X25519 and X448 curve support)

2.30 Release v02.16.01

Note:

This release fixes an issue introduced in v02.16.00 (see under 'APIs & enum/types Changes'). If you are using v02.16.00 please migrate to v02.16.01.

2.30.1 APIs & enum/types Changes

- Removed member `object_exist` from C structure `Se05xPolicy_t`. This solves a potential issue with the direct usage from the following APIs:
 - [Se05x_API_WriteECKey](#)
 - [Se05x_API_WriteRSAKey](#)
 - [Se05x_API_WriteSymmKey](#)
 - [Se05x_API_WriteBinary](#)
 - [Se05x_API_WritePCR](#)

The potential issue prevented by this change is as follows: In case `object_exist` was not properly initialized, the specific object policy passed as an argument for an object to be created would not be applied, instead the default policy would be applied.

The invocation of the above API's from the SSS layer has been modified accordingly.

2.30.2 Documentation Changes

- The enumerated type `SE05x_EDSignatureAlgo_t` contains a value `kSE05x_EDSignatureAlgo_ED25519_PH_SHA_512`. The mnemonic name of the value is misleading as it actually corresponds to the *Pure EDDSA algorithm* not the *Prehashed (PH) EDDSA algorithm*.

2.30.3 Other Miscellaneous Changes

- T1oI2C:
 - Fixed: potential null pointer dereference
 - Fixed: RSYNC _ + CRC error results in saving response to uninitialized buffer.
- Include pre-compiled a7x_vcom_to_* binaries for the iMX.RT1060 host platform (instead of iMX.RT1050) in the simw/binaries folder.
- Fixed issue with VCOM on RT1060

2.31 Release v02.16.00

2.31.1 File/Folder relocation

- `se05x.Inject_Certificate` renamed to `se05x.InjectCertificate`
- `se05x_minimal` renamed to `se05x_Minimal`
- `se05x.Get_Info` to `se05x.GetInfo`
- `se05x.Get_Certificate` to `se05x.GetCertificate`
- `ex_i2cMaster` to `se05x.I2cMaster`

2.31.2 New platform support

- Added support for MIMXRT1060

2.31.3 New feature support

- Added EdgeLock2go examples
- Montgomery X25519 curve support added for Openssl engine

2.31.4 SSSCLI / PyCLI Changes

- Added support to create object policy and attach to objects
- Sign and Verify input data restricted to file, Command line facility is removed

2.31.5 Examples / DEMO updates

- Added example for LPC55S based [Key Injection to PUF](#)
- Added example for LPC55S based [Key Rotation using PUF](#)
- Examples renamed
 - se05x_setAppletFeatures => se05x_SetAppletFeatures
 - se05x.Inject_Certificate => se05x.InjectCertificate
 - se05x_minimal => se05x_Minimal
 - ex_A71CHMain => a71ch_Main
 - se05x_Get_Info => se05x_GetInfo
 - se05x_Get_Certificate => se05x_GetCertificate
 - ex_i2cMaster => se05x_I2cMaster
 - ex_i2cMaster_with_Attestation => se05x_I2cMasterWithAttestation
 - cloud_demo_aws => cloud_aws
 - cloud_demo_gcp => cloud_gcp
 - cloud_demo_ibm_watson => cloud_ibm_watson
 - cloud_demo_azure => cloud_azure

2.31.6 Communication Layer Changes

- smComSerial for Linux PC: Do not translate CR (carriage return) into NL (newline) character when reading data

2.31.7 Other Miscellaneous Changes

- MIMXRT1050 support replaced with MIMXRT1060
- ImportExternalObjectCreate example is removed. ImportExternalObjectPrepare prepares and sends the object.

2.32 Release v02.15.00

Internal Milestone - Not published to NXP Website

2.32.1 Build system changes

- Changed FIPS Selection. Now use [PTMW_FIPS](#)-DSSS_HAVE_FIPS=ON is no longer valid. Use -DFIPS=SE050 to enable FIPS mode or -DFIPS=None to disable FIPS mode.

2.32.2 APIs & enum/types Changes

- Added support for extended applet features to be used with [Se05x_API_SetAppletFeatures\(\)](#).
- Updated [Se05x_API_SetAppletFeatures\(\)](#) and [sss_se05x_set_feature\(\)](#) signature. Defined new structure with applet features and extended features.
- Added `kAlgorithm_SSS_ECDA` in [sss_algorithm_t](#) for Barreto Naehrig curves.
- Refactored [sss_algorithm_t](#), added common algorithm `kAlgorithm_SSS_RSAES_PKCS1_V1_5`, removed SHA dependency.
- Added APIs `Se05x_API_WriteSymmKey_with_version()`,
`Se05x_API_WriteECKey_with_version()`, `Se05x_API_WriteRSAKey_with_version()` and
`Se05x_API_WriteBinary_with_version()` to create key with version information.
- Added API [Se05x_API_TLSCalculateRsaPreMasterSecret\(\)](#) to support RSA Pre-master secret with client version.
- Updated PCR functionality to not pass hash value from outside. Application should pass an input which will be hashed inside the applet.

2.32.3 Functional Changes

- Enable usage of usleep for clang.
- Moved PSA driver initialization to secure application on LPC55S.
- Added support for flash storage of PSA keystore on LPC55S. See [Managing KeyIDs](#).
- Added support for Platform SCP03 using PUF on LPC55S.

2.32.4 New feature support

- SEMS Lite added for applet install, upgrade, delete etc.
- Added support for AES GCM on AKM

2.32.5 Documentation Changes

- Added Documentation on ota-greengrass for Greengrass.
- Added documentation for SEMS Lite.

2.32.6 Examples / DEMO updates

- Updated example [Section 5.7.18Configuring Applet Features](#) to demonstrate use of extended features.
- Added attestation examples. [Section 5.2.1.8](#) and [Section 5.2.1.9](#)
- Added example to demonstrate `kAlgorithm_SSS_ECDA` signing operation.
- Added example for secure pairing on LPC55S
- Added examples for SEMS Lite
- Added examples for Perso scripts to demonstrate how to personalize the applet.

2.32.7 Other Miscellaneous Changes

- Files `simw-top/hostlib/hostLib/platform/imx/i2c_a7.c` and `simw-top/hostlib/hostLib/platform/rsp/i2c_a7.c` have been merged and the resulting I2C wrapper file is now located at `simw-top/hostlib/hostLib/platform/linux/i2c_a7.c`
- Memory leak fixes in T1oI2C layer
- Buffer size fixes, support for `CKA_LABEL`, `CKA_CERTIFICATE_TYPE` and `CKA_ALWAYS_AUTHENTICATE` in PKCS#11.

2.33 Release v02.14.00

2.33.1 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_MDFEV2 => ex_Ev2Prepare_Card
 - ex_prepare_se05x => ex_Ev2Prepare_se05x

2.33.2 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 4.7.1PTMW_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 4.7.2PTMW_SE05X_Ver](#) is introduced for future use.
- See [Section 4.7.11PTMW_SE05X_Auth](#)
 - FastSCP is now called ECKey.
 - AppletSCP is now called AESKey.

2.33.3 New platform support

- i.MX8 support added ([Setup i.MX 8MQuad - MCIMX8M-EVK](#))

2.33.4 APIs & enum/types Changes

- Use [se05x_API_ReadObject_W_Attst\(\)](#) 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_UM1225_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.

2.33.5 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 Key ID 00000001.

2.33.6 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

2.33.7 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.

2.33.8 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.

2.33.9 APIs & enum/types Changes

- Re-Wrote (internal) low level Tx/Rx APIs for APDU TxRx.
 - DoAPDUTxRx_s_Case2
 - DoAPDUTx_s_Case3
 - DoAPDUTxRx_s_Case4
 - 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.

2.33.10 Examples / DEMO updates

Updated Examples:

- [Section 5.7.2SE05X Get Info example](#) Updated to show CPLC data.

New Examples:

- [Section 5.7.11Import External Object Prepare](#)
import-external-obj-createimport-external-obj-create
- [Section 5.7.13Read object with Attestation](#)
- [Section 5.7.14SE05X Transport Lock example](#)
- [Section 5.7.15SE05X Transport UnLock example](#)
- [Section 5.7.16SE05X Timestamp](#)
- write-apdu-to-bufwrite-apdu-to-buf

2.33.11 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

2.33.12 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)

2.34 Release v02.12.05

Handle PC Linux for VCOM Interface.

2.35 Release v02.12.04

Minor touch ups.

2.36 Release v02.12.03

AKM: Fix processing of keys without read policy.

2.37 Release v02.12.02

2.37.1 AKM Specific changes

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

2.37.2 Example Updates

- Updated example [Section 5.2.1.2RSA Example](#) to also inject RSA key and run with FIPS compiled option.
- Added compile time check to skip examples [Section 5.2.1.4HKDF Example](#) and [Section 5.2.1.7ECDH Example](#) when compiled with FIPS

2.37.3 Other Miscellaneous Changes

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

2.38 Release v02.12.01

2.38.1 AKM Specific changes

- Modified SE policy of keymaster HAL in Android
- Added [Section 5.7.6SE05X Rotate PlatformSCP Keys Demo](#) to be build by Android build system.

2.38.2 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 5.7.6SE05X Rotate PlatformSCP Keys Demo](#) & its documentation.

2.38.3 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

2.39 Release v02.12.00

2.39.1 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 Keystream

2.39.2 SSSCLI / PyCLI Changes

- Updated AWS Provisioning certificate chain to remove Intermediate CA

2.39.3 Documentation Changes

- Added documentation for AWS Ease-of-Use

2.40 Release v02.11.03

2.40.1 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/a71x_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_a71x.c
```

2.40.2 New feature support

- WiFi and cloud demos support for LPC55S

2.40.3 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.

2.40.4 Documentation Changes

- Updated API usage description for T10I2C protocol stack

2.40.5 Other Miscellaneous Changes

- PKCS#11 Testbench Integration

2.41 Internal Release v02.11.01

2.41.1 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()`

2.41.2 Functional Changes

- Added support for jcpc 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 jcpc server V1.

2.41.3 New feature support

- Added support for AWS Greengrass core on Rpi.

2.41.4 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

2.41.5 Documentation Changes

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

2.41.6 Communication Layer Changes

- Removed unwanted buffer handling from T10I2C.

2.41.7 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 8.1 Introduction on OpenSSL engine](#)

2.41.8 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.

2.42 Release v02.11.00

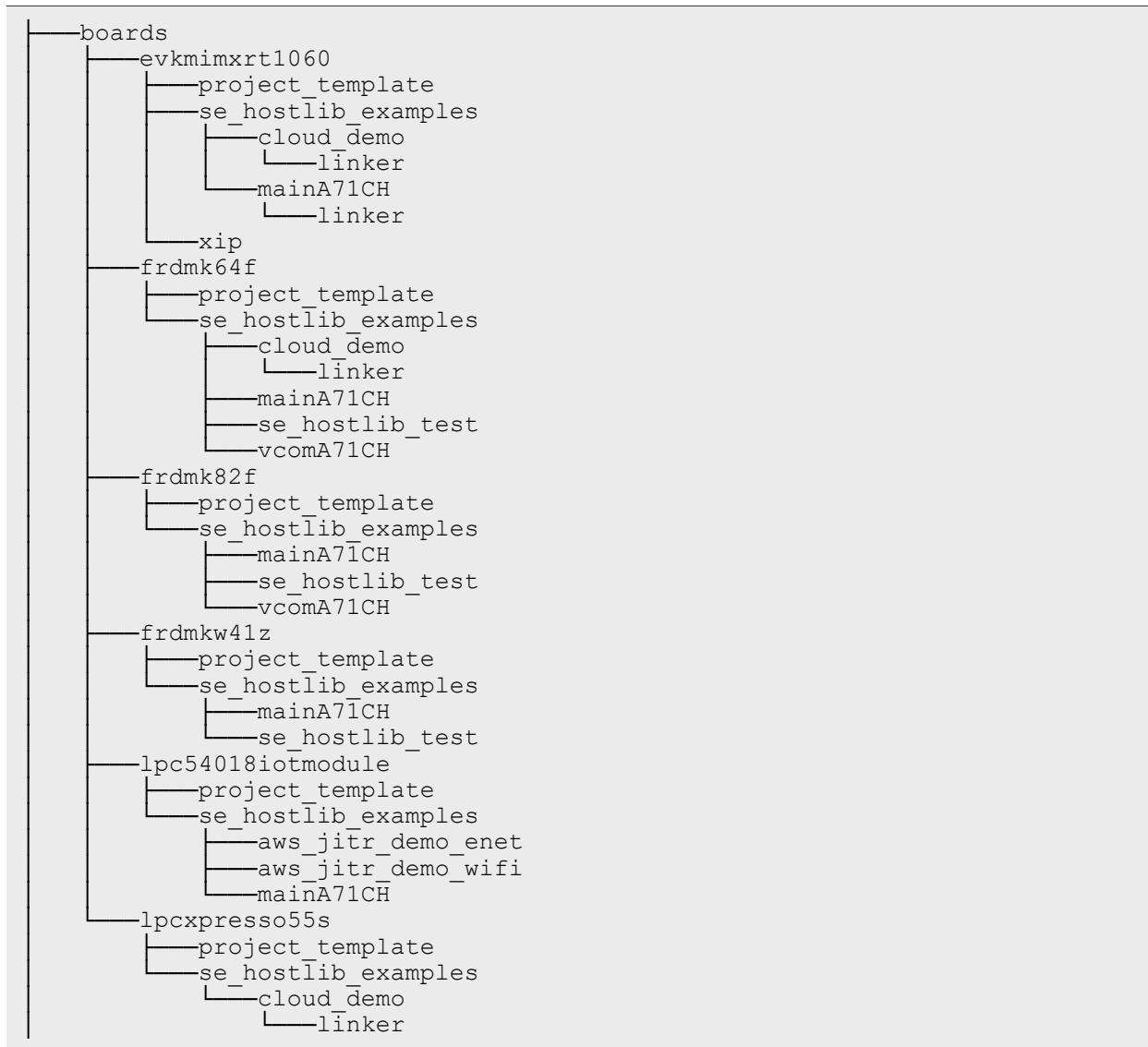
2.42.1 File/Folder relocation

boards folder has been moved from

`simw-top\ext\mcu-sdk`

to

simw-top\demos\ksdk\common



freertos folder moved from

simw-top\ext\freertos

to

simw-top\demos\ksdk\common\freertos

```
—freertos
|   FreeRTOSConfig.c
```

other boards folder has been moved from

simw-top\ext\boards

to

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



2.42.2 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`

2.42.3 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.

2.42.4 New platform support

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

2.42.5 Scripts and Build changes

- Added AOSP build support for Android keymaster.
- Added `Host=lpcxpresso55s_s` and `Host=lpcxpresso55s_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.

2.42.6 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

2.42.7 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

2.42.8 Communication Layer Changes

- Optimized T1oI2C transreceive 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.

2.42.9 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-RT1060 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

2.43 Release v02.10.00

2.43.1 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 keyVersioNo 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.

2.43.2 New platform support

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

2.43.3 Scripts and Build changes

- iMX RT 1050 drivers updated to SDK Release 2.6.1

2.43.4 SSSCLI / PyCLI Changes

- Added timeout mechanism in ssscli

2.43.5 Other Miscellaneous Changes

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

2.44 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

2.44.1 Middleware

- Refactor: Changed kSSS_KeyType_Certificate to kSSS_KeyType_Binary

2.44.2 CLI Tool

- Updated ssscli --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.

2.45 Release v02.07.00

2.45.1 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

2.45.2 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.

2.45.3 SSS APIs

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

2.45.4 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

2.45.5 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

2.46 Release v02.06.00

2.46.1 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

2.46.2 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

2.46.3 Development Platform

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

2.47 Release v02.05.00

Added Android Key Master.

2.48 Release v02.04.00

Added SE050 EAR CH and SE050 EAR CL Applet supports.

2.49 Release 02.03.00

Added support for SSS APIs.

3 Plug & Trust MW Stack

3.1 Features

Supported development platforms: [Development Platforms](#)

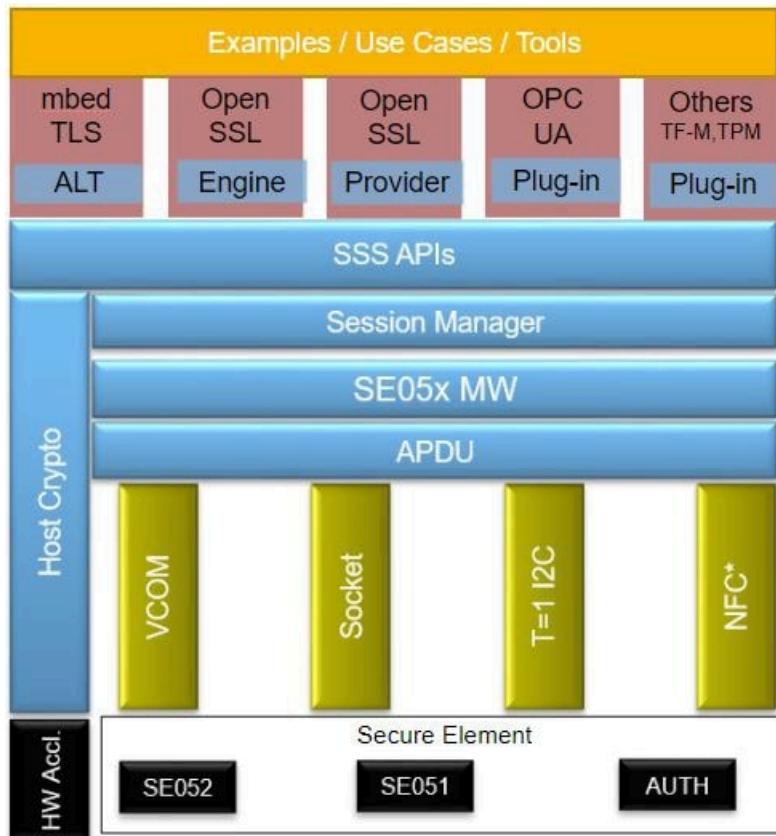
The Plug & Trust Middleware Stack supports:

- Directly accessible standard API/integration
 - mbed TLS
 - OpenSSL
 - Android KeyStore
 - PKCS #11
- Examples for quick integration
 - TLS examples
 - Cloud service on-boarding examples
- Integration with NXP EdgeLock 2GO cloud service

Note:

The supported features will vary according to the specific Secure Element used (e.g. SE050A, SE051C). Refer to the Secure Element specific Data Sheet (e.g. [SE050 Configuration](#)) to find out what functionality is available.

3.2 Plug & Trust MW : Block Diagram



3.3 SSS APIs

See also [Section 5.2.1 SSS API Examples](#)

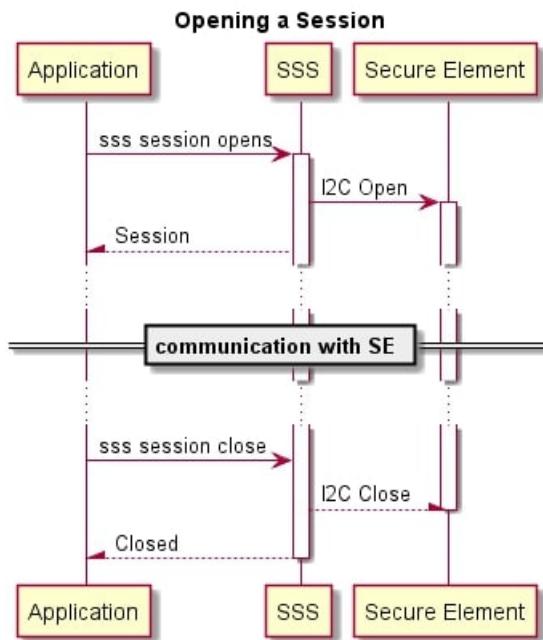
3.3.1 SSS: Introduction

SSS is an acronym for **S**ecure **S**ub **S**ystem

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\(\)](#)

3.3.2.1 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.

Note:

`sss_session_open()` must not be called concurrently from multiple threads. The application must ensure this.

3.3.2.1.1 SE05x Session

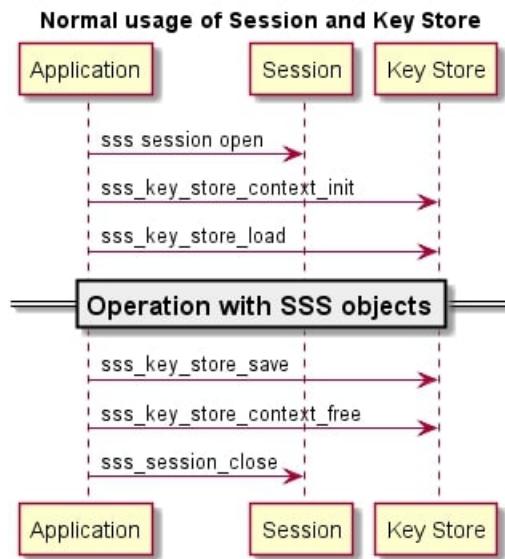
For example, a dedicated [SE_Connect_Ctx_t](#) is passed while opening a session to the SE05x Secure Element.

3.3.2.2 APIs

See [SSS Session types and APIs](#)

3.3.3 Key Store

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



3.3.3.1 APIs

See [SSS Keystore types and APIs](#)

3.3.3.2 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.

3.3.3.2.1 EC and RSA Key pair

When passing an EC / RSA 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.

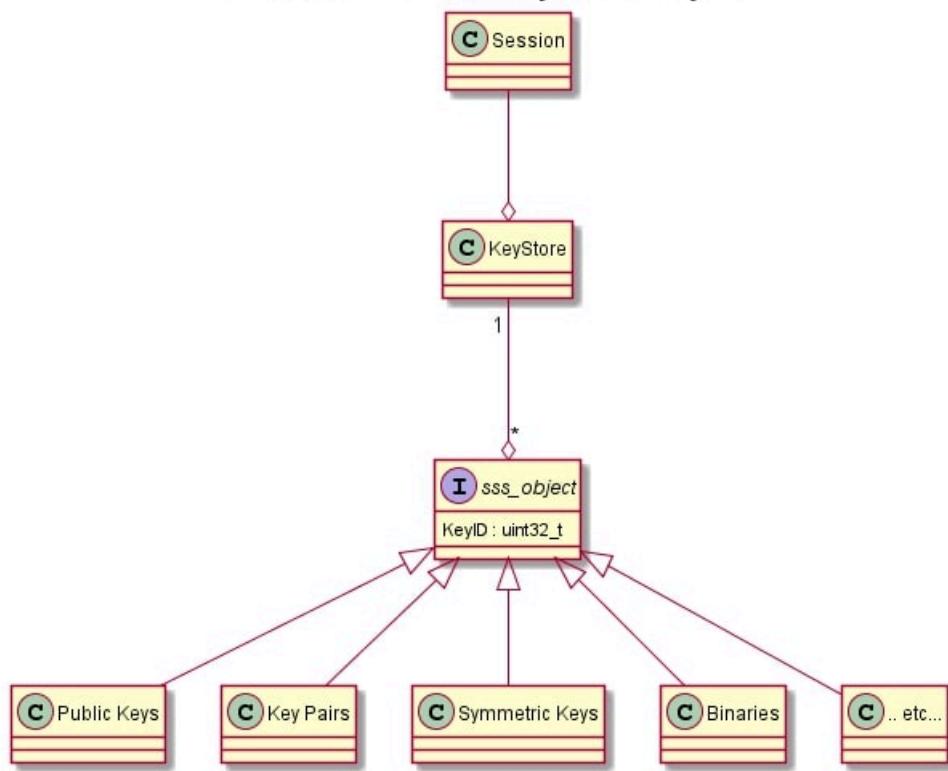
When retrieving an EC / RSA key pair as data argument from the `sss_key_store_get` API, the full key pair cannot be retrieved. Instead the public key value is returned. The public key is retrieved in ANSI X9.62 uncompressed format.

See [SSS api key format \(asymmetric keys\)](#)

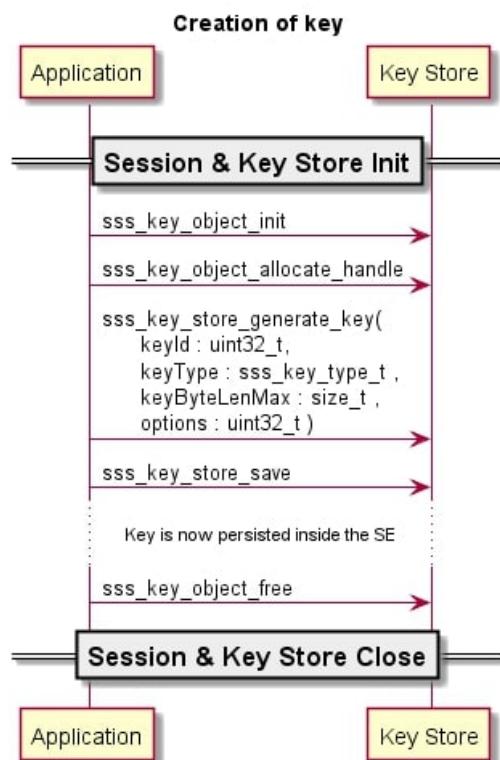
3.3.4 Key Object

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**3.3.4.1 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:

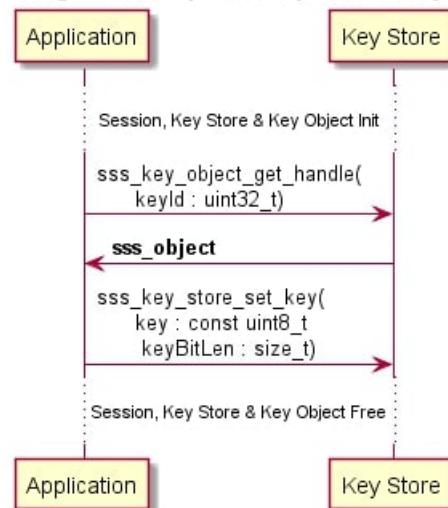
Policies

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

3.3.4.2 Change value of previously created Objects

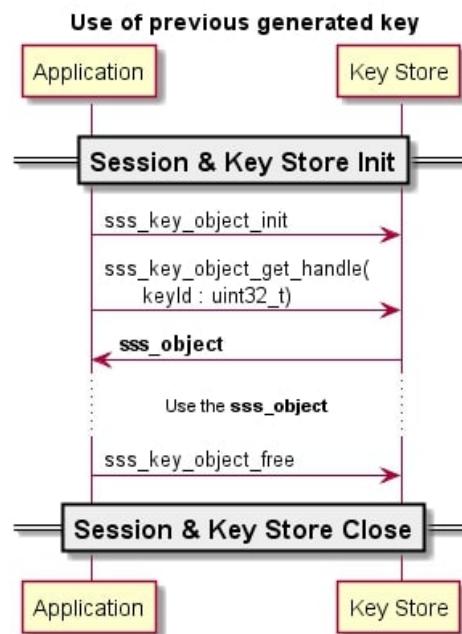
To create a key, the sequence of APIs looks as under:

Change value of previously created object



3.3.4.3 Use previously provisioned/created Keys/Objects

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



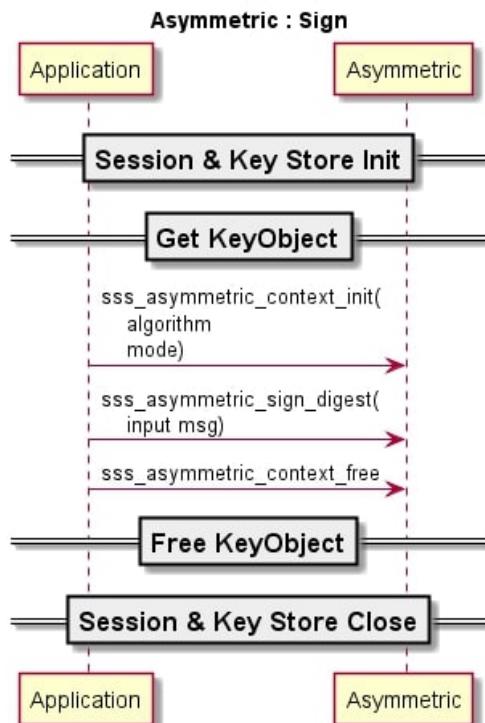
3.3.4.4 APIs

See [SSS KeyObject types and APIs](#)

3.3.5 Asymmetric

3.3.5.1 Sign

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

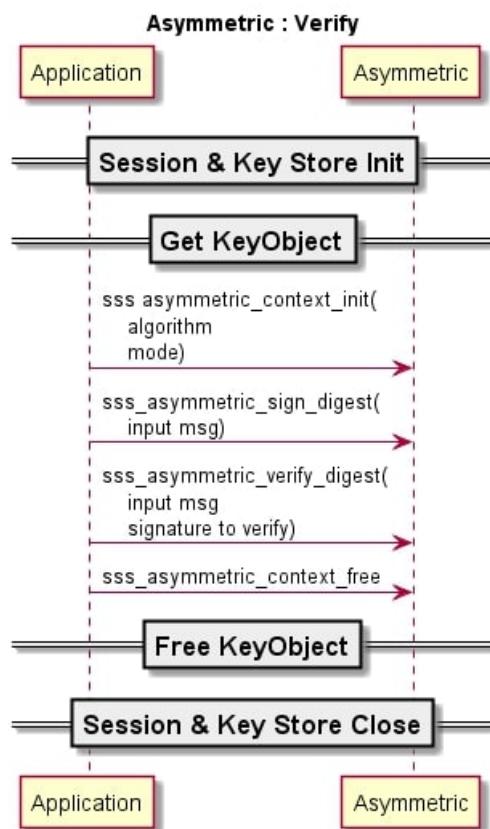


Note:

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.

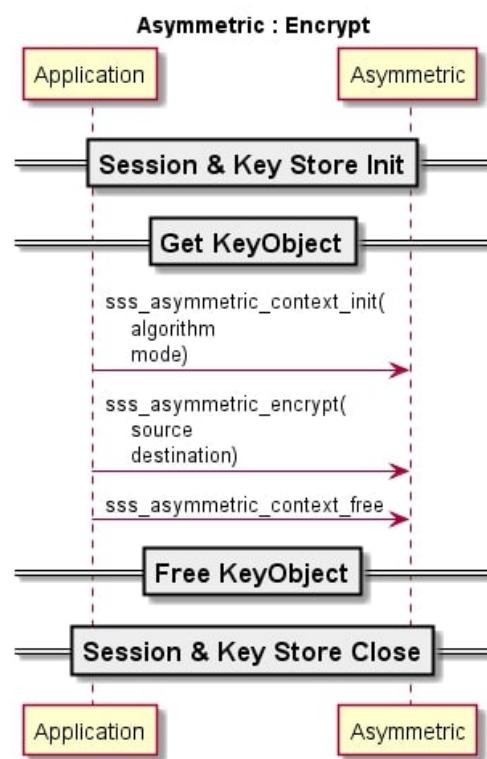
3.3.5.2 Verify

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



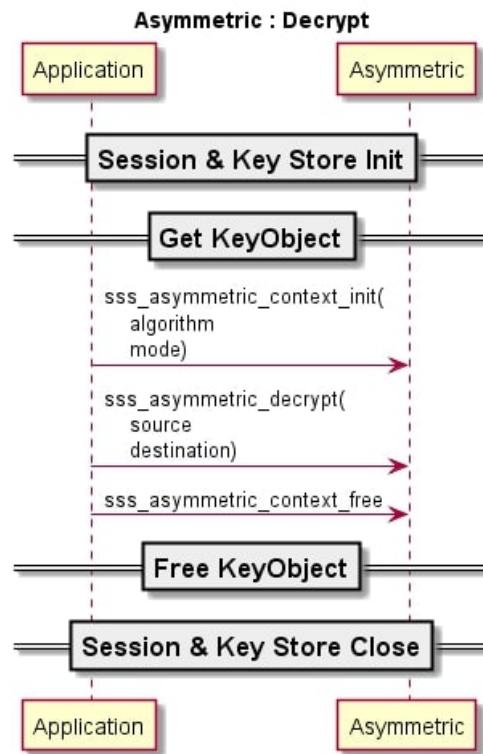
3.3.5.3 Encryption

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



3.3.5.4 Decryption

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



3.3.5.5 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`.

3.3.5.6 RSA Encryption algorithms supported

Supported rsa encyption / decryption algotithms - `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),
kAlgorithm_SSS_RSAES_PKCS1_V1_5           = SSS_ENUM_ALGORITHM(RSAES_PKCS1_V1_
5, 0x01),

```

3.3.5.7 RSA Signature algorithms supported

Supported rsa sign / verify algotithms - `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

3.3.5.8 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

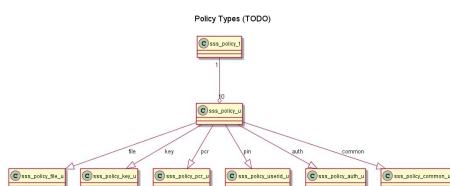
```
kAlgorithm_SSS_SHA1 = SSS_ENUM_ALGORITHM(SHA, 0x01),
kAlgorithm_SSS_SHA224 = SSS_ENUM_ALGORITHM(SHA, 0x02),
kAlgorithm_SSS_SHA256 = SSS_ENUM_ALGORITHM(SHA, 0x03),
kAlgorithm_SSS_SHA384 = SSS_ENUM_ALGORITHM(SHA, 0x04),
kAlgorithm_SSS_SHA512 = SSS_ENUM_ALGORITHM(SHA, 0x05),
```

3.3.5.9 APIs

See [SSS Asymmetric types and APIs](#)

3.3.6 Policies

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



3.3.6.1 Policies applicable to different objects

Object Type	Applet 3.x	Applet 6.x	Applet 7.x
common	forbid_All, can_Delete, req_Sm, req_pcr_val	forbid_All, can_Delete, req_Sm, req_pcr_val	forbid_All, can_Delete, req_Sm, req_pcr_val, can_Read, can_Write
symmetric objects	can_Sign, can_Verify, can_Encrypt, can_Decrypt, can_Import_Export, can_Wrap, can_Desfire_Auth, can_Desfire_Dump, can_KD, can_Write , can_Gen	can_Sign, can_Verify, can_Encrypt, can_Decrypt, can_Import_Export, forbid_Derived_Output, allow_kdf_ext_rnd, can_Wrap, can_Desfire_Auth, can_Desfire_Dump, can_KD, can_Write , can_Gen	can_Sign, can_Verify, can_Encrypt, can_Decrypt, can_Import_Export, forbid_Derived_Output, can_TLS_KDF, allow_kdf_ext_rnd, can_TLS_PMS_KD, can_HKDF, can_PBKDF, can_Wrap, can_Desfire_Auth, can_Desfire_Dump, can_Desfire_KD, forbid_external_iv, can_usage_hmac_pepper
Asymmetric objects	can_Sign, can_Verify, can_Encrypt, can_Decrypt, can_Import_Export, can_Gen, can_KA, can_Attest, can_Read, can_Write , can_KD, can_Wrap	can_Sign, can_Verify, can_Encrypt, can_Decrypt, can_Import_Export, forbid_Derived_Output, can_Gen, can_KA, can_Attest, can_Read, can_Write , can_KD, can_Wrap	can_Sign, can_Verify, can_Encrypt, can_Decrypt, can_Import_Export, forbid_Derived_Output, can_Gen, can_KA, can_Attest,
User Id	can_Write	can_Write	can_Write
File policy	can_Read, can_Write	can_Read, can_Write	can_Read, can_Write
Counter policy	can_Read, can_Write	can_Read, can_Write	can_Read, can_Write
PCR policy	can_Read, can_Write	can_Read, can_Write	can_Read, can_Write

Note:

1. **can_Read and can_Write** polices are moved from symmetric and asymmetric object policy to common policy in applet 7.x. **PLEASE UPDATE THE APPLICATIONS ACCORDINGLY.**
2. Invalid policies on objects will be rejected at SSS software layer and only result in warning message.

3.3.6.2 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*/
        .asymkey = {
            /*Policy for sign*/
            .can_Sign = allow_sign,
            /*Policy for verify*/
            .can_Verify = allow_verify,
            /*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,
    }
}
};%%CRLF
SP%/* 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,
        }
    }
};%%CRLF
SP%/* 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);

```

Note:

When creating a policy with `KPolicy_Common_PCR_Value`, `KPolicy_Desfire_Changekey_Auth_Id` and `KPolicy_Derive_Master_Key_Id`, set the policies to `sss_policy_t` variable in the following order always,

`'Common_PCR_Value'`, `'Desfire_Changekey_Auth_Id'`, `'Derive_Master_Key_Id'`

Example,

```

const sss_policy_u pcr_val_policy = {
    .type = KPolicy_Common_PCR_Value, .auth_obj_id = 0,
    .policy = { .common_pcr_value = {} } };

const sss_policy_u desfire_change_key_auth_id_policy = {
    .type = KPolicy_Desfire_Changekey_Auth_Id, .auth_obj_id = 0,
    .policy = { .desfire_auth_id = {} } };

```

```
const sss_policy_u master_key_id_policy = {
.type = KPolicy_Derive_Master_Key_Id, .auth_obj_id = 0,
.policy = { .master_key_id = {} } };

// create policy as,
sss_policy_t key_policy = { .nPolicies = 3,.policies = { &pcr_val_policy,
&desfire_change_key_auth_id_policy, &master_key_id_policy } };
// OR
sss_policy_t key_policy = { .nPolicies = 1,.policies =
{ &desfire_change_key_auth_id_policy } };
// OR
sss_policy_t key_policy = { .nPolicies = 2,.policies = { &pcr_val_policy,
&master_key_id_policy } };
```

3.3.6.3 APIs

See [SSS Policy types and APIs](#)

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.

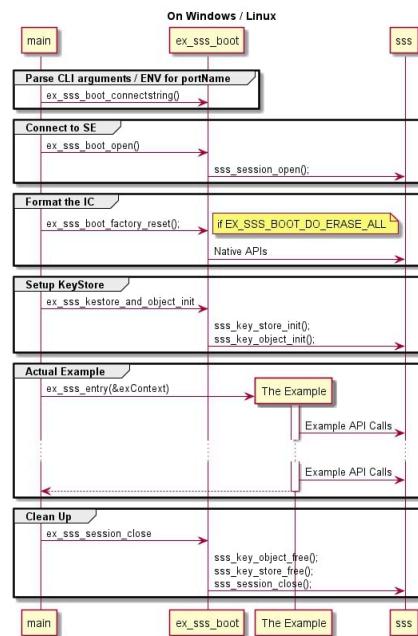
Some of the scenarios of boot up are:

3.3.7.1 Booting 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.

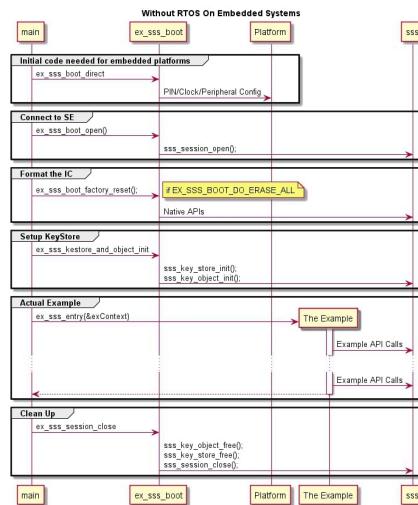
In such a system, examples also have access to command line arguments and environment variables.

Such a setup is mostly for testing and early prototyping.



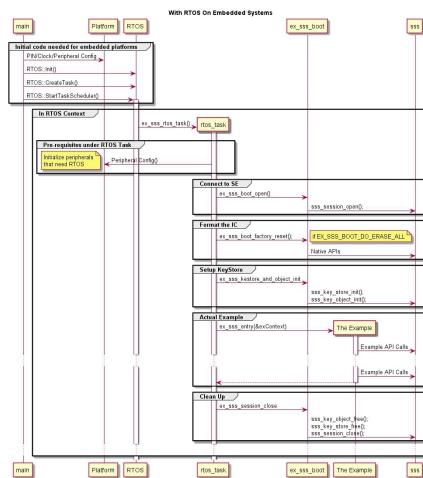
3.3.7.2 Booting from an embedded system, without any RTOS

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



3.3.7.3 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)

3.3.8.1 NIST-P, NIST-K and BRAINPOOL ECC Keys

When passing the key pair / public key, the key should be passed DER encoded using PKCS #8 or traditional OpenSSL format. When passing the private key alone, do not include the key header.

Example key formats for some keys are shown below -

NIST-P 256 Key pair (Traditional OpenSSL format):

```
$ dumpasn1 ecc_256R_1_prv_openssl.der
0 119: SEQUENCE {
2   1:   INTEGER 1
5  32:   OCTET STRING
:     2B 30 F6 68 F3 98 83 75 28 26 05 E2 EE 97 65 3A
:     41 2A 97 F8 0A 87 BC 7E 5F DD 5B D2 5C 5F 40 8E
39 10: [0] {
41  8:   OBJECT IDENTIFIER prime256v1 (1 2 840 10045 3 1 7)
:
51 68: [1] {
53 66:   BIT STRING
:     04 87 02 B7 5B 01 73 8A F0 5F 2D FF 17 85 37 1A
:     62 ED 6F C5 FD 7D BA 6F 14 45 05 43 23 3B 1B 06
:     59 46 2B B6 25 DD 7D D6 77 BB 35 F9 72 99 A3 6A
:     D4 33 B8 54 F6 93 3A CB 9F F6 47 DE D0 B6 9E 01
:     5D
:
:   }
:
: }
```

NIST-P 256 Key pair (PKCS8 format):

```
$ dumpasn1 ecc_256R_1_prv_pkcs8.der
0 135: SEQUENCE {
3   1:   INTEGER 0
6  19:   SEQUENCE {
8   7:     OBJECT IDENTIFIER ecPublicKey (1 2 840 10045 2 1)
17  8:     OBJECT IDENTIFIER prime256v1 (1 2 840 10045 3 1 7)
:
}
```

```

27 109: OCTET STRING, encapsulates {
29 107:   SEQUENCE {
31 1:     INTEGER 1
34 32:     OCTET STRING
:       2B 30 F6 68 F3 98 83 75 28 26 05 E2 EE 97 65 3A
:       41 2A 97 F8 0A 87 BC 7E 5F DD 5B D2 5C 5F 40 8E
68 68:   [1] {
70 66:     BIT STRING
:       04 87 02 B7 5B 01 73 8A F0 5F 2D FF 17 85 37 1A
:       62 ED 6F C5 FD 7D BA 6F 14 45 05 43 23 3B 1B 06
:       59 46 2B B6 25 DD 7D D6 77 BB 35 F9 72 99 A3 6A
:       D4 33 B8 54 F6 93 3A CB 9F F6 47 DE D0 B6 9E 01
:       5D
:     }
:   }
: }
:
```

NIST-P 256 Public key

```

$ dumpasn1 ecc_256R_1_pub.der
0 89: SEQUENCE {
2 19:   SEQUENCE {
4 7:     OBJECT IDENTIFIER ecPublicKey (1 2 840 10045 2 1)
13 8:     OBJECT IDENTIFIER prime256v1 (1 2 840 10045 3 1 7)
:   }
23 66:   BIT STRING
:     04 B9 46 AA E7 08 69 DA 27 2F 55 55 0A FA D9 42
:     F1 90 A4 0C 64 14 D3 78 94 74 AC 07 D5 F8 2C C9
:     53 05 F5 B1 4C 3E F7 0A 23 9A 3D 50 0D 90 92 D7
:     3B AB EA 12 23 FA A4 46 20 6F 22 4C CC E1 A5 1C
:     70
:   }
:
```

NIST-K 256 Key pair (Traditional OpenSSL format):

```

$ dumpasn1 ecc_256K_1_prv_openssl.der
0 116: SEQUENCE {
2 1:   INTEGER 1
5 32:   OCTET STRING
:     3F 0F 98 81 29 82 A7 6A AA 8A BA F1 7C 9C DC 7A
:     D8 9E 20 8A 5B 86 DB 58 AD 9C 1B 32 43 4A 26 4D
39 7:   [0] {
41 5:     OBJECT IDENTIFIER secp256k1 (1 3 132 0 10)
:   }
48 68:   [1] {
50 66:     BIT STRING
:       04 63 AF AF 56 B8 D2 84 CA 92 BE 88 97 E8 EA B5
:       EA A7 AC 30 94 CF 4A 5A 4B 97 67 1B B1 3A B2 85
:       8C 95 FE 2C D9 4D 00 C7 EF FF 29 B5 CD A2 46 2D
:       4C CD BD D1 CB 14 49 A2 10 26 C4 35 B9 ED DD 37
:       D7
:     }
:   }
:
```

NIST-K 256 Key pair (PKCS8 format):

```

$ dumpasn1 ecc_256K_1_prv_pkcs8.der
0 132: SEQUENCE {
:
```

```
3   1:   INTEGER 0
6  16: SEQUENCE {
8   7:   OBJECT IDENTIFIER ecPublicKey (1 2 840 10045 2 1)
17  5:   OBJECT IDENTIFIER secp256k1 (1 3 132 0 10)
:   }
24 109: OCTET STRING, encapsulates {
26 107:   SEQUENCE {
28   1:     INTEGER 1
31  32:     OCTET STRING
:       3F 0F 98 81 29 82 A7 6A AA 8A BA F1 7C 9C DC 7A
:       D8 9E 20 8A 5B 86 DB 58 AD 9C 1B 32 43 4A 26 4D
65  68:   [1] {
67  66:     BIT STRING
:       04 63 AF AF 56 B8 D2 84 CA 92 BE 88 97 E8 EA B5
:       EA A7 AC 30 94 CF 4A 5A 4B 97 67 1B B1 3A B2 85
:       8C 95 FE 2C D9 4D 00 C7 EF FF 29 B5 CD A2 46 2D
:       4C CD BD D1 CB 14 49 A2 10 26 C4 35 B9 ED DD 37
:       D7
:     }
:   }
: }
```

NIST-K 256 Public key

```
$ dumpasn1 ecc_256K_1_pub.der
0 86: SEQUENCE {
2 16:   SEQUENCE {
4   7:     OBJECT IDENTIFIER ecPublicKey (1 2 840 10045 2 1)
13  5:     OBJECT IDENTIFIER secp256k1 (1 3 132 0 10)
:   }
20  66:   BIT STRING
:     04 3B 98 11 47 B9 7C 9B 39 62 2B 54 42 F6 0E 16
:     B1 57 5E 17 23 08 DE 16 02 55 D7 EB 09 53 24 15
:     9C B0 6A 49 EB 59 EF 5C AE 2A 31 48 CD 88 F5 ED
:     EF 12 24 3C 36 EF C2 3F DA B1 AE EC E3 EE DD EA
:     DB
: }
```

BrainPool 256 Key pair (Traditional OpenSSL format):

```
$ dumpasn1 ecc_256B_1_prv_openssl.der
0 120: SEQUENCE {
2   1:   INTEGER 1
5  32:   OCTET STRING
:     55 27 E9 FA 47 E8 58 B3 B0 CF D1 0C 4C 03 2A 80
:     1E AB 3F C4 68 A7 2B B3 F8 E4 DC BD 47 E7 4E C1
39 11:   [0] {
41   9:     OBJECT IDENTIFIER brainpoolP256r1 (1 3 36 3 3 2 8 1 1 7)
:   }
52  68:   [1] {
54  66:     BIT STRING
:       04 7D A6 C9 C2 CC 5A 4A 18 C5 7B 92 1F A1 96 3B
:       49 3D 2A 97 BB B5 3B 1D 09 ED F9 51 85 61 A4 F9
:       C7 19 7F 00 6D A3 9B 61 C2 09 AA 65 EF 65 1D 2E
:       B4 38 C6 F0 29 D3 5D DC 98 D1 19 75 D3 7B C3 00
:       E6
:     }
:   }
: }
```

BrainPool 256 Key pair (PKCS8 format):

```
$ dumpasn1 ecc_256B_1_prv_pkcs8.der
0 136: SEQUENCE {
3   1:   INTEGER 0
6  20:   SEQUENCE {
8    7:     OBJECT IDENTIFIER ecPublicKey (1 2 840 10045 2 1)
17   9:     OBJECT IDENTIFIER brainpoolP256r1 (1 3 36 3 3 2 8 1 1 7)
      }
28 109:   OCTET STRING, encapsulates {
30 107:     SEQUENCE {
32   1:       INTEGER 1
35  32:       OCTET STRING
      :         55 27 E9 FA 47 E8 58 B3 B0 CF D1 0C 4C 03 2A 80
      :         1E AB 3F C4 68 A7 2B B3 F8 E4 DC BD 47 E7 4E C1
69  68:       [1] {
71  66:         BIT STRING
      :           04 7D A6 C9 C2 CC 5A 4A 18 C5 7B 92 1F A1 96 3B
      :           49 3D 2A 97 BB B5 3B 1D 09 ED F9 51 85 61 A4 F9
      :           C7 19 7F 00 6D A3 9B 61 C2 09 AA 65 EF 65 1D 2E
      :           B4 38 C6 F0 29 D3 5D DC 98 D1 19 75 D3 7B C3 00
      :           E6
      :
      }
      :
      }
      :
      }
      :
      }
```

BrainPool-256 Public key

```
$ dumpasn1 ecc_256B_1_pub.der
0 90: SEQUENCE {
2 20:   SEQUENCE {
4   7:     OBJECT IDENTIFIER ecPublicKey (1 2 840 10045 2 1)
13   9:     OBJECT IDENTIFIER brainpoolP256r1 (1 3 36 3 3 2 8 1 1 7)
      }
24  66:   BIT STRING
      :     04 7D A6 C9 C2 CC 5A 4A 18 C5 7B 92 1F A1 96 3B
      :     49 3D 2A 97 BB B5 3B 1D 09 ED F9 51 85 61 A4 F9
      :     C7 19 7F 00 6D A3 9B 61 C2 09 AA 65 EF 65 1D 2E
      :     B4 38 C6 F0 29 D3 5D DC 98 D1 19 75 D3 7B C3 00
      :     E6
      :
      }
```

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.

3.3.8.2 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
      :
      }

$ 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
      :
```

The X25519 keypair from the example above can be created with the `asn1parse` OpenSSL tool. The command line invocation is as follows:

```
openssl asn1parse -genconf key_config.txt -out X25519_keypair.der
```

The file **key_config.txt** contains the different ASN.1 components and values of the keypair to create. The resulting DER representation of the keypair is stored in the file **X25519_keypair.der**. The contents of the **key_config.txt** file is as follows:

```
asn1 = SEQUENCE:seq_section
[seq_section]

field1 = INTEGER:1
field2 = SEQUENCE:seq_section_oid
```

```

field3 =
OCTWRAP, FORMAT:HEX, OCT:585DB1E3500B7124F6B1E14183549312F44B0CA344F752A18A122FE7DAD9CE52
field4 =
IMP:1C, INT:0x00A28E04FF1CDC1C3D60910FBC98EF01BF9F0F69C0B7EF7061353462F30628C729
[seq_section_oid]

field1 = OID:X25519

```

Note:

Additional information on the asn1parse command and the key configuration format is available under the manpages of respectively asn1parse and ASN1_generate_nconf

3.3.8.2.1 Set Edwards and Montgomery keys using pycli

On generating keys using openssl, pycli tool can be used to set edwards and montgomery keys.

Example - Set key pair as:

```
`openssl genpkey -algorithm ed25519 -outform PEM -out test_ed25519.pem`  
`sscli set ecc pair <KEYID> test_ed25519.pem`
```

Set public key as:

```
`openssl pkey -in test_ed25519.pem -out test_ed25519_pub.pem -pubout`  
`sscli set ecc pub <KEYID> test_ed25519_pub.pem`
```

3.3.8.3 RSA keys

When passing the key pair / public key, the key should be passed DER encoded using PKCS #8 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.

Example key format for RSA-2048 key pair is shown below -

RSA-2048 Key Pair (Traditional OpenSSL format)

```
$ dumpasn1 rsa_2048_1_prv_openssl.der
0 1187: SEQUENCE {
 4   1:   INTEGER 0
 7 257:   INTEGER
    :     00 91 C9 82 8D 32 D6 39 F5 B2 59 15 19 4B B8 F8
    :     71 DE 62 7A C3 E8 B9 7E 99 06 23 0C D3 1B 4B 45
    :     09 8F DD F3 69 FD 6B 79 90 0A BB 4A E9 6E 43 0C
    :     D2 4E 63 09 3C 90 45 7D D6 8A C5 54 17 D0 3D FE
    :     77 20 D3 1E 8B E3 8A B4 7C E0 06 18 3D 16 B4 C2
    :     08 47 D0 4D 08 27 EB AF 12 9B 59 2B 8F EC 6D F1
    :     ED 1C 7A 4A 5A 99 D4 DF 40 54 F3 0B 72 5D 31 E6
    :     54 70 8E 0F 55 8C 32 08 F1 46 EB 39 00 0D E5 C9
    :       [ Another 129 bytes skipped ]
268   3:   INTEGER 65537
273 256:   INTEGER
    :     17 B0 C9 24 1C 83 E9 D4 EE 8A 57 09 39 20 11 FB
    :     75 B9 C7 72 5E 37 5A 86 65 14 12 B4 46 DB 58 98
    :     D6 5B 53 D5 6A F1 93 F1 0A 2A BD 0B 29 8D AA CA
    :     8D 78 BE DA 56 7A D1 C3 FD E1 55 7F 83 29 B7 D8
```

```

:
: 1A FF 99 17 54 14 1E 58 01 3D 96 F8 6D 0A AA FC
:
: 96 7D 97 D9 AB 47 10 E2 F3 57 59 0D 61 21 CD 59
:
: 0C 87 7B 88 0A 8C 7D 9F A2 23 AC 13 5B 12 A3 4D
:
: 20 77 42 B0 52 56 6F D3 E9 84 35 A0 8D 21 9A 14
:
[ Another 128 bytes skipped ]
533 129: INTEGER
:
: 00 C1 FA D9 5B 58 84 98 79 52 80 A5 BC 36 3C 32
:
: 7D 4F 15 51 8B A7 1B 32 5E 0F 52 55 A3 75 0B BA
:
: F1 B5 14 E5 5B 66 33 40 03 B2 0D 71 56 79 2A 4F
:
: 1F 28 81 21 96 8F FF 28 4E F2 9C 45 BB 2F 6D BC
:
: 6E DC B6 AB F1 57 B0 6A 50 64 37 86 26 A9 C1 D5
:
: 59 39 4F 00 4B 6E C0 0C 07 AF DC 12 70 0F CB B8
:
: F1 E4 00 4F 04 62 1A 86 C1 A8 45 3A 7A AB D1 EF
:
: 86 EF BC 7A 6A 53 7E BF 2B 5C 5A 33 B8 2B 60 CC
:
: 5B
665 129: INTEGER
:
: 00 C0 66 1C 7C E9 4B 20 5E 3A E4 35 EE 74 F2 0C
:
: A8 F8 4F EF E8 46 22 BE A2 13 36 EC A9 C4 50 30
:
: 1D 8E DC 1D DA D0 BF A1 7E 85 DE 6B 82 EC EE 3B
:
: 4A F6 C4 6F 81 23 B1 84 09 C8 E3 1D F4 DC 3B 48
:
: EF 15 9D 05 95 14 04 30 B5 7F 80 FE FE D3 3C 75
:
: 75 AD F2 F0 90 5F A6 0C 28 F3 65 48 59 82 7C 49
:
: 38 B4 F6 AB 8F 87 39 20 2D D7 3E 2E 00 D4 47 3F
:
: 28 9F 5C 9F F4 1F FC 7D 67 C7 46 00 6E 99 04 CD
:
: 65
797 129: INTEGER
:
: 00 A4 5D D6 47 9C 9D DE 45 0F 2F 8B 40 0C 04 BE
:
: 13 88 2B 5C 49 A5 73 5A 1E 71 85 26 A3 B6 CE 15
:
: BE 31 DE 5E EA 2F 93 45 AE DB F4 A0 10 D1 E2 93
:
: E0 A7 05 A4 5C 5B EF AD 4C 18 2F 6A B6 CD DD 82
:
: 49 BE 23 DB 56 49 23 67 32 6F 78 CC E7 7D F8 8C
:
: BB 69 E0 13 33 D7 C8 4B 69 48 0E 86 61 06 41 6D
:
: 99 29 C5 49 2F 41 A1 90 86 0F FB 79 2D F0 E1 96
:
: C1 13 EA F5 1F 9B 58 4E CC 83 18 BB B2 56 AF 52
:
: F9
929 128: INTEGER
:
: 1A 22 44 A0 5A F8 0F 6F 7D 44 5E 67 03 8F 95 54
:
: A6 56 05 5A 61 9C 7A 94 7D 53 AA 95 EC CA 8F 9E
:
: 94 37 25 FF 00 F7 E6 B1 CE F1 45 5D 45 5D 9E C4
:
: 31 FC C0 C0 A3 DE 8A F6 E1 48 A8 5B 08 47 2D 42
:
: FC 86 95 A9 88 4C 81 69 45 E6 79 BC 97 68 D0 F3
:
: A9 2B 24 AE 17 AF F0 5A E7 A4 CC 4D 0C 42 61 97
:
: C8 4C F1 44 CF B3 5C C1 9D 49 1E EA 91 EB 13 93
:
: 2B 02 63 DF BF 30 86 C0 3F FB 2C 37 D9 A5 23 59
:
1060 128: INTEGER
:
: 0E AD DC 66 09 36 F6 A9 55 28 FD C3 47 6D B9 1D
:
: 88 15 41 1B A2 23 5A B7 81 4E C8 90 A2 51 50 7E
:
: D9 DF 83 0B 64 DF A9 5F AF 20 DA 65 BC C8 74 F9
:
: E2 00 6E 15 1D 8E DF 47 C6 84 BF 96 F7 44 C9 38
:
: FC B8 70 B5 ED C2 3C BC B6 B3 5B 54 C3 85 66 42
:
: B7 5A 70 99 37 A8 55 C0 F6 96 F5 EF F2 34 8D 3F
:
: A3 56 2E 5D A4 97 79 00 4D 28 3E 35 03 19 41 89
:
: A3 96 A5 45 37 90 84 C1 F5 6E 75 3F 33 24 44 FF
:
: }

```

RSA-2048 Key Pair (PKCS8 format)

```

$ dumpasn1 rsa_2048_1_prv_pkcs8.der
0 1213: SEQUENCE {
4   1:   INTEGER 0

```

```
7   13: SEQUENCE {
9     9:   OBJECT IDENTIFIER rsaEncryption (1 2 840 113549 1 1 1)
20    0:   NULL
      :
22 1191: OCTET STRING, encapsulates {
26 1187:   SEQUENCE {
30    1:     INTEGER 0
33  257:     INTEGER
      :
      :       00 91 C9 82 8D 32 D6 39 F5 B2 59 15 19 4B B8 F8
      :       71 DE 62 7A C3 E8 B9 7E 99 06 23 0C D3 1B 4B 45
      :       09 8F DD F3 69 FD 6B 79 90 0A BB 4A E9 6E 43 0C
      :       D2 4E 63 09 3C 90 45 7D D6 8A C5 54 17 D0 3D FE
      :       77 20 D3 1E 8B E3 8A B4 7C E0 06 18 3D 16 B4 C2
      :       08 47 D0 4D 08 27 EB AF 12 9B 59 2B 8F EC 6D F1
      :       ED 1C 7A 4A 5A 99 D4 DF 40 54 F3 0B 72 5D 31 E6
      :       54 70 8E 0F 55 8C 32 08 F1 46 EB 39 00 0D E5 C9
      :
      :       [ Another 129 bytes skipped ]
294    3:   INTEGER 65537
299  256:   INTEGER
      :
      :       17 B0 C9 24 1C 83 E9 D4 EE 8A 57 09 39 20 11 FB
      :       75 B9 C7 72 5E 37 5A 86 65 14 12 B4 46 DB 58 98
      :       D6 5B 53 D5 6A F1 93 F1 0A 2A BD 0B 29 8D AA CA
      :       8D 78 BE DA 56 7A D1 C3 FD E1 55 7F 83 29 B7 D8
      :       1A FF 99 17 54 14 1E 58 01 3D 96 F8 6D 0A AA FC
      :       96 7D 97 D9 AB 47 10 E2 F3 57 59 0D 61 21 CD 59
      :       0C 87 7B 88 0A 8C 7D 9F A2 23 AC 13 5B 12 A3 4D
      :       20 77 42 B0 52 56 6F D3 E9 84 35 A0 8D 21 9A 14
      :
      :       [ Another 128 bytes skipped ]
559  129:   INTEGER
      :
      :       00 C1 FA D9 5B 58 84 98 79 52 80 A5 BC 36 3C 32
      :       7D 4F 15 51 8B A7 1B 32 5E 0F 52 55 A3 75 0B BA
      :       F1 B5 14 E5 5B 66 33 40 03 B2 0D 71 56 79 2A 4F
      :       1F 28 81 21 96 8F FF 28 4E F2 9C 45 BB 2F 6D BC
      :       6E DC B6 AB F1 57 B0 6A 50 64 37 86 26 A9 C1 D5
      :       59 39 4F 00 4B 6E C0 0C 07 AF DC 12 70 0F CB B8
      :       F1 E4 00 4F 04 62 1A 86 C1 A8 45 3A 7A AB D1 EF
      :       86 EF BC 7A 6A 53 7E BF 2B 5C 5A 33 B8 2B 60 CC
      :
      :       5B
691  129:   INTEGER
      :
      :       00 C0 66 1C 7C E9 4B 20 5E 3A E4 35 EE 74 F2 0C
      :       A8 F8 4F EF E8 46 22 BE A2 13 36 EC A9 C4 50 30
      :       1D 8E DC 1D DA D0 BF A1 7E 85 DE 6B 82 EC EE 3B
      :       4A F6 C4 6F 81 23 B1 84 09 C8 E3 1D F4 DC 3B 48
      :       EF 15 9D 05 95 14 04 30 B5 7F 80 FE FE D3 3C 75
      :       75 AD F2 F0 90 5F A6 0C 28 F3 65 48 59 82 7C 49
      :       38 B4 F6 AB 8F 87 39 20 2D D7 3E 2E 00 D4 47 3F
      :       28 9F 5C 9F F4 1F FC 7D 67 C7 46 00 6E 99 04 CD
      :
      :       65
823  129:   INTEGER
      :
      :       00 A4 5D D6 47 9C 9D DE 45 0F 2F 8B 40 0C 04 BE
      :       13 88 2B 5C 49 A5 73 5A 1E 71 85 26 A3 B6 CE 15
      :       BE 31 DE 5E EA 2F 93 45 AE DB F4 A0 10 D1 E2 93
      :       E0 A7 05 A4 5C 5B EF AD 4C 18 2F 6A B6 CD DD 82
      :       49 BE 23 DB 56 49 23 67 32 6F 78 CC E7 7D F8 8C
      :       BB 69 E0 13 33 D7 C8 4B 69 48 0E 86 61 06 41 6D
      :       99 29 C5 49 2F 41 A1 90 86 0F FB 79 2D F0 E1 96
      :       C1 13 EA F5 1F 9B 58 4E CC 83 18 BB B2 56 AF 52
      :
      :       F9
955  128:   INTEGER
      :
      :       1A 22 44 A0 5A F8 0F 6F 7D 44 5E 67 03 8F 95 54
```

```

:
A6 56 05 5A 61 9C 7A 94 7D 53 AA 95 EC CA 8F 9E
:
94 37 25 FF 00 F7 E6 B1 CE F1 45 5D 45 5D 9E C4
:
31 FC C0 C0 A3 DE 8A F6 E1 48 A8 5B 08 47 2D 42
:
FC 86 95 A9 88 4C 81 69 45 E6 79 BC 97 68 D0 F3
:
A9 2B 24 AE 17 AF F0 5A E7 A4 CC 4D 0C 42 61 97
:
C8 4C F1 44 CF B3 5C C1 9D 49 1E EA 91 EB 13 93
:
2B 02 63 DF BF 30 86 C0 3F FB 2C 37 D9 A5 23 59
1086 128: INTEGER
:
0E AD DC 66 09 36 F6 A9 55 28 FD C3 47 6D B9 1D
:
88 15 41 1B A2 23 5A B7 81 4E C8 90 A2 51 50 7E
:
D9 DF 83 0B 64 DF A9 5F AF 20 DA 65 BC C8 74 F9
:
E2 00 6E 15 1D 8E DF 47 C6 84 BF 96 F7 44 C9 38
:
FC B8 70 B5 ED C2 3C BC B6 B3 5B 54 C3 85 66 42
:
B7 5A 70 99 37 A8 55 C0 F6 96 F5 EF F2 34 8D 3F
:
A3 56 2E 5D A4 97 79 00 4D 28 3E 35 03 19 41 89
:
A3 96 A5 45 37 90 84 C1 F5 6E 75 3F 33 24 44 FF
:
}
:
}
:
}

```

RSA-2048 Public Key

```

$ dumpasn1 rsa_2048_1_pub.der
0 290: SEQUENCE {
4 13:  SEQUENCE {
6 9:   OBJECT IDENTIFIER rsaEncryption (1 2 840 113549 1 1 1)
17 0:   NULL
:
19 271: BIT STRING, encapsulates {
24 266:   SEQUENCE {
28 257:     INTEGER
:
00 BF 4C DD D8 6C 13 A2 B2 EB E3 81 53 0C BC 12
:
19 46 7C 7E 05 27 EF 0A 73 60 39 90 3D 11 00 D7
:
31 D3 F8 D5 DE 90 92 EF D1 75 B9 0D 3E 46 AC 46
:
EE 57 12 1A 07 3B 7C 63 08 9B B2 3B B4 0C A8 30
:
99 16 AD 96 10 FE 1C 2E 30 94 EE 92 A1 6F D8 28
:
D4 72 80 24 77 CF 50 8A 38 72 DA 3F 9E BD 53 E2
:
4D FB D3 08 5E 93 F5 49 55 6F AC 03 CF 06 1E 79
:
DD 86 BB CC F0 4B 5B FC 5D 35 39 77 BA 7C AE 93
:
[ Another 129 bytes skipped ]
289 3:   INTEGER 65537
:
}
:
}
:
}

```

3.3.9 List of all SSS APIs and structures

- [SSS Enums and Types](#)
- [SSS Session types and APIs](#)
- [SSS Keystore types and APIs](#)
- [SSS KeyObject types and APIs](#)
- [SSS Symmetric types and APIs](#)
- [SSS Asymmetric types and APIs](#)
- [SSS RNG types and APIs](#)
- [SSS Digest types and APIs](#)
- [SSS MAC types and APIs](#)

- [SSS Key derivation types and APIs](#)
- [SSS AEAD types and APIs](#)
- [SSS Tunnel types and APIs](#)
- [SSS Policy types and APIs](#)
- [SSS Str log types and APIs](#)

3.3.9.1 SSS Enums and Types

groupsss_types

SSS Types.

Defines

SSS_ENUM(GROUP, INDEX)

Helper macro to set enum value

Enums

enum **sss_access_permission_t**

Permissions of an object

Values:

enumerator **kAccessPermission_SSS_Read**

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.

enumerator **kAccessPermission_SSS_Write**

Can change the value of an object

enumerator **kAccessPermission_SSS_Use**

Can use an object

enumerator **kAccessPermission_SSS_Delete**

Can delete an object

enumerator **kAccessPermission_SSS_ChangeAttributes**

Can change permissions applicable to an object

enumerator **kAccessPermission_SSS_All_Permission**

Bitwise OR of all sss_access_permission.

enum **sss_algorithm_t**

Cryptographic algorithm to be applied

Values:

enumerator **kAlgorithm_None**

enumerator **kAlgorithm_SSS_AES_ECB**

enumerator **kAlgorithm_SSS_AES_CBC**

enumerator **kAlgorithm_SSS_AES_CTR**

enumerator **kAlgorithm_SSS_AES_GCM**

enumerator **kAlgorithm_SSS_AES_CCM**

enumerator **kAlgorithm_SSS_AES_GCM_INT_IV**

```
enumerator kAlgorithm_SSS_AES_CTR_INT_IV
enumerator kAlgorithm_SSS_AES_CCM_INT_IV
enumerator kAlgorithm_SSS_CHACHA_POLY
enumerator kAlgorithm_SSS_DES_ECB
enumerator kAlgorithm_SSS_DES_CBC
enumerator kAlgorithm_SSS_DES_CBC_ISO9797_M1
enumerator kAlgorithm_SSS_DES_CBC_ISO9797_M2
enumerator kAlgorithm_SSS_DES3_ECB
enumerator kAlgorithm_SSS_DES3_CBC
enumerator kAlgorithm_SSS_DES3_CBC_ISO9797_M1
enumerator kAlgorithm_SSS_DES3_CBC_ISO9797_M2
enumerator kAlgorithm_SSS_SHA1
enumerator kAlgorithm_SSS_SHA224
enumerator kAlgorithm_SSS_SHA256
enumerator kAlgorithm_SSS_SHA384
enumerator kAlgorithm_SSS_SHA512
enumerator kAlgorithm_SSS_CMAC_AES
enumerator kAlgorithm_SSS_HMAC_SHA1
enumerator kAlgorithm_SSS_HMAC_SHA224
enumerator kAlgorithm_SSS_HMAC_SHA256
enumerator kAlgorithm_SSS_HMAC_SHA384
enumerator kAlgorithm_SSS_HMAC_SHA512
enumerator kAlgorithm_SSS_DES_CMAC8
enumerator kAlgorithm_SSS_DH
enumerator kAlgorithm_SSS_ECDH
enumerator kAlgorithm_SSS_DSA_SHA1
enumerator kAlgorithm_SSS_DSA_SHA224
enumerator kAlgorithm_SSS_DSA_SHA256
enumerator kAlgorithm_SSS_RSASSA_PKCS1_V1_5_NO_HASH
enumerator kAlgorithm_SSS_RSASSA_PKCS1_V1_5_SHA1
enumerator kAlgorithm_SSS_RSASSA_PKCS1_V1_5_SHA224
enumerator kAlgorithm_SSS_RSASSA_PKCS1_V1_5_SHA256
enumerator kAlgorithm_SSS_RSASSA_PKCS1_V1_5_SHA384
enumerator kAlgorithm_SSS_RSASSA_PKCS1_V1_5_SHA512
enumerator kAlgorithm_SSS_RSASSA_PKCS1_PSS_MGF1_SHA1
```

```
enumerator kAlgorithm_SSS_RSASSA_PKCS1_PSS_MGF1_SHA224
enumerator kAlgorithm_SSS_RSASSA_PKCS1_PSS_MGF1_SHA256
enumerator kAlgorithm_SSS_RSASSA_PKCS1_PSS_MGF1_SHA384
enumerator kAlgorithm_SSS_RSASSA_PKCS1_PSS_MGF1_SHA512
enumerator kAlgorithm_SSS_RSAES_PKCS1_OAEP_SHA1
enumerator kAlgorithm_SSS_RSAES_PKCS1_OAEP_SHA224
enumerator kAlgorithm_SSS_RSAES_PKCS1_OAEP_SHA256
enumerator kAlgorithm_SSS_RSAES_PKCS1_OAEP_SHA384
enumerator kAlgorithm_SSS_RSAES_PKCS1_OAEP_SHA512
enumerator kAlgorithm_SSS_RSAES_PKCS1_V1_5
enumerator kAlgorithm_SSS_RSASSA_NO_PADDING
enumerator kAlgorithm_SSS_ECDSA_SHA1
enumerator kAlgorithm_SSS_ECDSA_SHA224
enumerator kAlgorithm_SSS_ECDSA_SHA256
enumerator kAlgorithm_SSS_ECDSA_SHA384
enumerator kAlgorithm_SSS_ECDSA_SHA512

enum sss_cipher_type_t
```

For all cipher types, key bit length is provided at the time key is inserted/generated
Values:

```
enumerator kSSS_CipherType_NONE
enumerator kSSS_CipherType_AES
enumerator kSSS_CipherType_DES
enumerator kSSS_CipherType_CMAC
enumerator kSSS_CipherType_HMAC
enumerator kSSS_CipherType_MAC
enumerator kSSS_CipherType_RSA
enumerator kSSS_CipherType_RSA_CRT
    RSA RAW format
enumerator kSSS_CipherType_EC_NIST_P
    RSA CRT format
enumerator kSSS_CipherType_EC_NIST_K
    Keys Part of NIST-P Family
enumerator kSSS_CipherType_EC_MONTGOMERY
    Keys Part of NIST-K Family Montgomery Key,
enumerator kSSS_CipherType_EC_TWISTED_ED
    twisted Edwards form elliptic curve public key
```

enumerator **kSSS_CipherType_EC_BRAINPOOL**
Brainpool form elliptic curve public key

enumerator **kSSS_CipherType_UserId**

enumerator **kSSS_CipherType_Certificate**
Use kSSS_CipherType_Binary to store Certificate

enumerator **kSSS_CipherType_Binary**

enumerator **kSSS_CipherType_Count**

enumerator **kSSS_CipherType_PCR**

enumerator **kSSS_CipherType_ReservedPin**

enum **sss_connection_type_t**
Destintion connection type
Values:

enumerator **kSSS_ConnectionType_Plain**

enumerator **kSSS_ConnectionType_Password**

enumerator **kSSS_ConnectionType_Encrypted**

enum **sss_key_object_mode_t**
Persistent / Non persistent mode of a key
Values:

enumerator **kKeyObject_Mode_None**
kKeyObject_Mode_None

enumerator **kKeyObject_Mode_Persistent**
Key object will be persisted in memory and will retain it's value after a closed session

enumerator **kKeyObject_Mode_Transient**
Key Object will be stored in RAM. It will lose it's contents after a session is closed

enum **sss_key_part_t**
Part of a key
Values:

enumerator **kSSS_KeyPart_NONE**

enumerator **kSSS_KeyPart_Default**
Applicable where we have UserID, Binary Files, Certificates, Symmetric Keys, PCR, HMAC-key, counter

enumerator **kSSS_KeyPart_Public**
Public part of asymmetric key

enumerator **kSSS_KeyPart_Private**
Private only part of asymmetric key

enumerator **kSSS_KeyPart_Pair**
Both, public and private part of asymmetric key

enum **sss_mode_t**
High level algorihtmic operations.
Augmented by [sss_algorithm_t](#)

Values:

enumerator **kMode_SSS_Encrypt**

Encrypt.

enumerator **kMode_SSS_Decrypt**

Decrypt.

enumerator **kMode_SSS_Sign**

Sign.

enumerator **kMode_SSS_Verify**

Verify.

enumerator **kMode_SSS_ComputeSharedSecret**

enumerator **kMode_SSS_Digest**

Message Digest.

enumerator **kMode_SSS_Mac**

Message Authentication Code.

enumerator **kMode_SSS_HKDF_ExpandOnly**

HKDF Expand Only (RFC 5869)

enumerator **kMode_SSS_HKDF_ExtractExpand**

HKDF Extract and Expand (RFC 5869)

enumerator **kMode_SSS_Mac_Validate**

MAC Validate.

enum **sss_status_t**

Status of the SSS APIs

Values:

enumerator **kStatus_SSS_Success**

Operation was successful

enumerator **kStatus_SSS_Fail**

Operation failed

enumerator **kStatus_SSS_InvalidArgument**

Operation not performed because some of the passed parameters were found inappropriate

enumerator **kStatus_SSS_ResourceBusy**

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.

enumerator **kStatus_SSS_ApduThroughputError**

APDU Throughput error

enum **sss_type_t**

Cryptographic sub system

Values:

enumerator **kType_SSS_SubSystem_NONE**

enumerator **kType_SSS_Software**

Software based

enumerator **kType_SSS_mbedTLS**

enumerator **kType_SSS_OpenSSL**

enumerator **kType_SSS_HW**

 HOST HW Based

 enumerator **kType_SSS_SECO**

 enumerator **kType_SSS_Isolated_HW**

 Isolated HW

 enumerator **kType_SSS_Sentinel**

 enumerator **kType_SSS_Sentinel200**

 enumerator **kType_SSS_Sentinel300**

 enumerator **kType_SSS_Sentinel400**

 enumerator **kType_SSS_Sentinel500**

 enumerator **kType_SSS_SecureElement**

 Secure Element

 enumerator **kType_SSS_SE_A71CH**

 To connect to <https://www.nxp.com/products/:A71CH>

 enumerator **kType_SSS_SE_A71CL**

 enumerator **kType_SSS_SE_SE05x**

 To connect to <https://www.nxp.com/products/:SE050>

 enumerator **kType_SSS_SubSystem_LAST**

struct **sss_ecc_point_t**

#include <fsl_sss_api.h>

XY Co-ordinates for ECC Curves

Public Members

uint8_t ***X**
 X Point

uint8_t ***Y**
 Y Point

struct **ssscgfp_group_t**

#include <fsl_sss_api.h>

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
```

3.3.9.2 SSS Session types and APIs

groupsss_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:

enumerator **kSSS_SessionProp_au8_NA**

Invalid

enumerator **kSSS_SessionProp_szName**

Name of the product, string

enumerator **kSSS_SessionProp_UID**

Unique Identifier

enumerator **kSSS_SessionProp_au8_Optional_Start**

Optional Properties Start

enumerator **kSSS_SessionProp_au8_Proprietary_Start**

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:

enumerator **kSSS_SessionProp_u32_NA**

Invalid

enumerator **kSSS_SessionProp_VerMaj**

Major version

enumerator **kSSS_SessionProp_VerMin**

Minor Version

enumerator **kSSS_SessionProp_VerDev**

Development Version

enumerator **kSSS_SessionProp_UIDLen**
enumerator **kSSS_SessionProp_u32_Optional_Start**
 Optional Properties Start
enumerator **kSSS_KeyStoreProp_FreeMem_Persistent**
 How much persistent memory is free
enumerator **kSSS_KeyStoreProp_FreeMem_Transient**
 How much transient memory is free
enumerator **kSSS_SessionProp_u32_Proprietary_Start**
 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 application has to call [sss_session_create](#) before calling [sss_session_open](#).

Parameters

- **session** – [inout] Pointer to session context
- **subsystem** – [in] See [sss_session_open](#)
- **application_id** – [in] See [sss_session_open](#)
- **connection_type** – [in] See [sss_session_open](#)
- **connectionData** – [in] 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 resources 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.

Note: [sss_session_open\(\)](#) must not be called concurrently from multiple threads. The application must ensure this.

Parameters

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

Returns

status

[**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.

Parameters

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

Returns

[**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](#)

Parameters

- **session** – [in] Session context
- **property** – [in] Value that is part of [sss_session_prop_u32_t](#)
- **pValue** – [out]

Returns

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*)) + (7 * sizeof(void*)) + 32)]

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

3.3.9.3 SSS Keystore types and 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:

enumerator kSSS_KeyStoreProp_au8_Optional_Start

Optional Properties Start

enum sss_tunnel_dest_t

Entity on the other side of the tunnel

Values:

enumerator kSSS_TunnelDest_None

Default value

enumerator 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, a new empty key store is created and initialized. Key store context structure is updated with actual information.

Parameters

- **keyStore** – [out] 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

- **keyStore** – [out] 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.

Parameters

- **keyStore** – The key store
- **keyObject** – The key object to be deleted

Returns

The sss status.

`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.

Parameters

- **keyStore** – The key store
- **keyObject** – The key object to be locked / frozen.

Returns

The sss status.

`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

Parameters

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

Returns

The sss status.

`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.

Parameters

- **keyStore** – Key store context
- **keyObject** – Reference to a key and its 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

Returns

```
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*)) + 32)]`

`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.

3.3.9.4 SSS KeyObject types and APIs

groupsss_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.

Parameters

- **keyObject** – **[inout]** 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 = Persistant Key (Default) or Transient Key. See [sss_key_object_mode_t](#)

Returns

Status of object allocation.

`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

Parameters

- **keyObject** – Object
- **access** – What is permitted

Returns

`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 @ref sss_key_object_allocate_handle.%CRLF

SP% After calling this API, Ideally keyObject should become equivlant to as set after the calling of @ref sss_key_object_allocate_handle api.

Parameters

- **keyObject** – The key object
- **keyId** – [in] The key identifier

Returns

The sss status.

`sss_status_t sss_key_object_get_purpose(sss_object_t *keyObject, sss_mode_t *purpose)`

Check what is purpose restrictions on an object

Parameters

- **keyObject** – Object to be checked
- **purpose** – Know what is permitted.

Returns

`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.

Parameters

- **keyObject** –
- **keyStore** –

Return values

- **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.

Returns

Status of the operation

`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*)) + 32)]

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 referenced from [sss_key_part_t](#)

3.3.9.5 SSS Symmetric types and APIs

group **sss_crypto_symmetric**

Symmetric cryptographic operations like AES / DES / etc.

Functions

```
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.

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.
- **initialCounter** – **[inout]** Input counter (Always 16 bytes) (updates on return). When using internal IV algorithms (only encrypt) for SE051, initial Counter buffer will be filled with generated Initial counter.
- **lastEncryptedCounter** – **[out]** Output cipher of last counter, for chained CTR calls. NULL can be passed if chained calls are not used.
- **szLeft** – **[out]** Output number of bytes in left unused in lastEncryptedCounter block. NULL can be passed if chained calls are not used.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

```
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.

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.
- **destLen** – **[inout]** Length of output data in bytes. Buffer length on entry, reflects actual output size on return.

Return values

- **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.

Returns

Status of the operation

```
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.

Parameters

- **context** – Pointer to symmetric crypto context.
- **iv** – Buffer containing the symmetric operation Initialization Vector. When using internal IV algorithms (only encrypt) for SE051, iv buffer will be filled with generated Initialization Vector.
- **ivLen** – Length of the Initialization Vector in bytes.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

`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.

Parameters

- **context** – Pointer to symmetric crypto context.
- **iv** – Buffer containing the symmetric operation Initialization Vector. When using internal IV algorithms (only encrypt) for SE051, iv buffer will be filled with generated Initialization Vector.
- **ivLen** – Length of the Initialization Vector in bytes.
- **srcData** – Buffer containing the input data (block aligned).
- **destData** – Buffer containing the output data.
- **dataLen** – Size of input and output data buffer in bytes.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

`sss_status_t sss_cipher_one_go_v2(sss_symmetric_t *context, uint8_t *iv, size_t ivLen, const uint8_t *srcData, const size_t srcLen, 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.

Parameters

- **context** – Pointer to symmetric crypto context.
- **iv** – Buffer containing the symmetric operation Initialization Vector. When using internal IV algorithms (only encrypt) for SE051, iv buffer will be filled with generated Initialization Vector.
- **ivLen** – Length of the Initialization Vector in bytes.
- **srcData** – Buffer containing the input data (block aligned).
- **srcLen** – Length of buffer srcData.
- **destData** – Buffer containing the output data.
- **dataLen** – Pointer to Size of buffer destData in bytes.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

```
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\(\)](#).

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.
- **destLen** – **[inout]** Length of the output data in bytes. Buffer length on entry, reflects actual output size on return.

Return values

- **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.

Returns

Status of the operation

```
void sss_symmetric_context_free(sss_symmetric_t *context)
```

Symmetric context release. The function frees symmetric context.

Parameters

context – Pointer to symmetric crypto context.

```
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.

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 values

- **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.

Returns

Status of the operation

```
struct sss_symmetric_t
```

```
#include <fsl_sss_api.h>
```

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 + 32)]
```

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.

3.3.9.6 SSS Asymmetric types and 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.

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 values

- **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.

Returns

Status of the operation

```
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.

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 values

- **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.

Returns

Status of the operation

```
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.

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 values

- **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.

Returns

Status of the operation

```
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.

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 values

- **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.

Returns

Status of the operation

`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.

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 values

- **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.

Returns

Status of the operation

3.3.9.7 SSS RNG types and APIs

groupsss_rng

Functions

`sss_status_t sss_rng_context_free(sss_rng_context_t *context)`

free random generator context.

Parameters

context – generator context.

Returns

status

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

Parameters

- **session** – Session context.
- **context** – random generator context.

Returns

status

```
sss_status_t sss_rng_get_random(sss_rng_context_t *context, uint8_t *random_data, size_t dataLen)
```

Generate random number.

Parameters

- **context** – random generator context.
- **random_data** – buffer to hold random data.
- **dataLen** – required random number length

Returns

status

```
struct sss_rng_context_t  
#include <fsl_sss_api.h>  
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*)) + 32)]
```

```
sss_session_t *session
```

Pointer to the session

3.3.9.8 SSS Digest types and APIs

```
groupsss_crypto_digest
```

Functions

```
void sss_digest_context_free(sss_digest_t *context)
```

Digest context release. The function frees digest context.

Parameters

```
context – Pointer to digest context.
```

```
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.

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 values

- **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.

Returns

Status of the operation

`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.

Parameters

- **context** – Pointer to digest context.
- **digest** – Output message digest
- **digestLen** – Message digest byte length

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

`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.

Parameters

context – Pointer to digest context.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

`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.

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 values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

[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.

Parameters

- **context** – Pointer to digest context.
- **message** – Buffer with a message chunk.
- **messageLen** – Length of the input buffer in bytes.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

struct sss_digest_t

```
#include <fsl_sss_api.h>
```

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*)) + 32)]
```

[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.

3.3.9.9 SSS MAC types and APIs

groupsss_crypto_mac**Functions****[void sss_mac_context_free\(sss_mac_t *context\)](#)**

MAC context release. The function frees mac context.

Parameters

context – Pointer to mac context.

[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.

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 values

- **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.

Returns

Status of the operation

[`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.

Parameters

- **context** – Pointer to mac context.
- **mac** – Output message MAC
- **macLen** – Computed MAC byte length

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

[`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.

Parameters

context – Pointer to mac context.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

[`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.

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 values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

`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.

Parameters

- **context** – Pointer to mac context.
- **message** – Buffer with a message chunk.
- **messageLen** – Length of the input buffer in bytes.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

struct **sss_mac_t**

#include <fsl_sss_api.h>

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 (kMode_SSS_Mac)

`sss_session_t *session`

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

3.3.9.10 SSS Key derivation types and APIs

group **sss_crypto_derive_key**

Derive Key.

Operations that use multiple keys from multiple security systems and derive and output key, e.g. Diffie Hellman exchange.

Functions

`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.

`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.

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 values

- `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.

Returns

Status of the operation

`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.

Parameters

- `context` – Pointer to derive key context.
- `otherPartyKeyObject` – Public key of the other party in the Diffie-Hellmann algorithm
- `derivedKeyObject` – **[inout]** Reference to a derived key

Return values

- `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.

Returns

Status of the operation

`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.

Deprecated:

Please use `sss_derive_key_one_go` instead

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.
- **derivedKeyObject** – **[inout]** Reference to a derived key
- **deriveDataLen** – Requested length of output
- **hkdfOutput** – Output buffer containing key derivation output
- **hkdfOutputLen** – Output containing length of hkdfOutput

Return values

- **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.

Returns

Status of the operation

`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.

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.
- **derivedKeyObject** – **[inout]** Reference to a derived key
- **deriveDataLen** – **[in]** Expected length of derived key.

Return values

- **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.

Returns

Status of the operation

`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.

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.
- **info** – [in] Input data buffer, typically with some fixed info.
- **infoLen** – [in] Length of info buffer in bytes.
- **derivedKeyObject** – Reference to a derived key
- **deriveDataLen** – [in] The derive data length

Return values

- **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.

Returns

Status of the operation

struct sss_derive_key_t

#include <fsl_sss_api.h>

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*)) + 32)]

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

3.3.9.11 SSS AEAD types and APIs

group**sss_crypto_aead**

Authenticated Encryption with Additional Data.

Functions

void sss_aead_context_free(sss_aead_t *context)

AEAD context release. The function frees aead context.

Parameters

context – Pointer to aead context.

sss_status_t sss_aead_context_init(sss_aead_t *context, sss_session_t *session, sss_object_t *keyObject, sss_algorithm_t algorithm, sss_mode_t mode)

AEAD context init. The function initializes aead context with initial values.

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 values

- **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.

Returns

Status of the operation

```
sss_status_t sss_aead_init(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).

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.
- **destLen** – **[inout]** 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 values

- **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.

Returns

Status of the operation

```
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)
```

AEAD init. The function starts the aead operation.

Parameters

- **context** – Pointer to aead crypto context.
- **nonce** – The operation nonce or IV. When using internal IV algorithms (only encrypt) for SE051, iv buffer will be filled with generated Initialization Vector.
- **nonceLen** – The length of nonce in bytes. For AES-GCM it must be ≥ 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 values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

```
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.

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. When using internal IV algorithms (only encrypt) for SE051, iv buffer will be filled with generated Initialization Vector.
- **nonceLen** – The length of nonce in bytes. For AES-GCM it must be ≥ 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 (Default value is 16 bytes). 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 values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

```
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.

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.
- **destLen** – **[inout]** Length of the output data in bytes. Buffer length on entry, reflects actual output size on return.

Return values

- **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.

Returns

Status of the operation

[**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.

Parameters

- **context** – Pointer to aead crypto context
- **aadData** – Input buffer containing the chunk of AAD
- **aadDataLen** – Length of the AAD data in bytes.

Return values

- **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.

Returns

Status of the operation

struct **sss_aead_t**

#include <fsl_sss_api.h>

Authenticated Encryption with Additional Data.

Public Members

[**sss_algorithm_t algorithm**](#)

Algorithm to be used

uint8_t data[(0 + (5 * sizeof(void*)) + (6 * sizeof(int)) + (5 * sizeof(void*)) + 32)]

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**](#)

High level operation (encrypt/decrypt)

[sss_session_t](#) *session

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

3.3.9.12 SSS Tunnel types and APIs

groupsss_crypto_tunnel

Tunnel session.

Functions

```
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

- **context** – **[inout]** 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.

```
void sss_tunnel_context_free(sss_tunnel_t *context)
```

Destructor for the tunnelling service context.

Parameters

context – **[out]** Pointer to tunnel context.

```
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

- **context** – **[out]** Pointer to tunnel context. Tunnel context is updated on function return.
- **session** – Pointer to session this tunnelling service belongs to.

```
struct sss_tunnel_t
```

```
#include <fsl_sss_api.h>
```

Tunneling

Used for communication via another system.

Public Members

```
uint8_t data[(0 + (1 * sizeof(void*)) + (2 * sizeof(int)) + (2 * sizeof(void*)) + 32)]
```

```
struct sss_tunnel_t::[anonymous] extension
```

Reserved memory for implementation specific extension

[sss_session_t](#) *session

Pointer to the session

`uint32_t tunnelType`

Tunnel to which Applet (Currently unused)

3.3.9.13 SSS Policy types and APIs

`groupsss_policy`

Policies to restrict and control sessions and objects.

Enums

enum `sss_policy_type_u`

Type of policy

Values:

enumerator `KPolicy_None`

No policy applied

enumerator `KPolicy_Session`

Policy related to session.

See Also:

[`sss_policy_session_u`](#)

enumerator `KPolicy_Sym_Key`

Policy related to key.

See Also:

[`sss_policy_key_u`](#)

enumerator `KPolicy_Asym_Key`

enumerator `KPolicy_UserID`

enumerator `KPolicy_File`

enumerator `KPolicy_Counter`

enumerator `KPolicy_PCR`

enumerator `KPolicy_Common`

enumerator `KPolicy_Common_PCR_Value`

enumerator `KPolicy_Desfire_Changekey_Auth_Id`

enumerator `KPolicy_Derive_Master_Key_Id`

enumerator `KPolicy_Internal_Sign`

struct `sss_policy_asym_key_u`

`#include <fsi/_sss_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_sss_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 can_Read
    Allow to read the object
uint8_t can_Write
```

Allow to write the object

uint8_t forbid_All

Forbid all operations

uint8_t req_pcr_val

Require PCR value

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_desfire_changekey_authId_value_u

#include <fsl_sss_policy.h>

DESFire ChangeKey - authentication key identifier.

Public Members

uint32_t desfire_authId

DESFire authentication object ID

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_internal_sign_tbs_value_u

#include <fsl_sss_policy.h>

Allow internal sign.

Public Members

uint32_t tbsItemList_KeyId

identifier of the tbsItemList Secure Object

struct sss_policy_key_drv_master_keyid_value_u

#include <fsl_sss_policy.h>

Key Derive - Master key identifier.

Public Members

```
uint32_t masterKeyId
```

Master key ID

```
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 allow_kdf_ext_rnd
```

Allow kdf(prf) external random

```
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_Desfire_KD
```

Allow Desfire key derivation

uint8_t can_Encrypt

Allow encryption

uint8_t can_Gen

Allow to (re)generate the object

uint8_t can_HKDF

Allow HKDF

uint8_t can_Import_Export

Allow to imported or exported

uint8_t can_KA

Allow key agreement. Only for SE051H, key agreement is applicable for symm objects

uint8_t can_KD

Allow key derivation

uint8_t can_PBKDF

Allow PBKDF

uint8_t can_Sign

Allow signature generation

uint8_t can_TLS_KDF

Allow TLS PRF key derivation

uint8_t can_TLS_PMS_KD

Allow TLS PMS key derivation

uint8_t can_usage_hmac_pepper

Allow usage as hmac pepper

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

uint8_t forbid_external_iv

Forbid External iv

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_desfire_changekey_authId_value_u desfire_auth_id
```

```
sss_policy_file_u file
```

```
sss_policy_key_dry_master_keyid_value_u master_key_id
```

```
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_internal_sign_tbs_value_u tbsItemList
```

```
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

3.3.9.14 SSS Str log types and APIs

```
group sss_str_log
```

Functions

```
const char *sss_cipher_type_sz(sss_cipher_type_t cipher_type)
```

Returns string error code for [sss_cipher_type_t](#).

Parameters

`cipher_type` – [in] See [sss_cipher_type_t](#)

Returns

String conversion of `cipher_type` to String.

```
const char *sss_status_sz(sss_status_t status)
```

Returns string error code for `sss_status_t`.

Parameters

`status` – [in] See `sss_status_t`

Returns

String conversion of `status` to String.

3.3.10 List of all SSS SE05x APIs and structures

- [SSS SE05x Enums and Types](#)
- [SSS SE05x Session types and APIs](#)
- [SSS SE05x Keystore types and APIs](#)
- [SSS SE05x KeyObject types and APIs](#)
- [SSS SE05x Symmetric types and APIs](#)
- [SSS SE05x Asymmetric types and APIs](#)
- [SSS SE05x RNG types and APIs](#)
- [SSS SE05x Digest types and APIs](#)
- [SSS SE05x MAC types and APIs](#)
- [SSS SE05x Key derivation types and APIs](#)
- [SSS SE05x AEAD types and APIs](#)
- [SSS SE05x Tunnel types and APIs](#)
- [SSS SE05x I2C Master types and APIs](#)
- [SSS SE05x Attestation types and APIs](#)
- [SSS SE05x Other types and APIs](#)

3.3.10.1 SSS SE05x Enums and Types

groupsss_sw_se05x

Manage session.

Defines

`se05x_auth_context_t`

deprecated : Used only for backwards compatibility

`SE05x_Connect_Ctx_t`

deprecated : Used only for backwards compatibility

`SSS_AEAD_TYPE_IS_SE05X(context)`

Are we using SE05X as crypto subsystem?

`SSS_ASYMMETRIC_TYPE_IS_SE05X(context)`

Are we using SE05X as crypto subsystem?

`SSS_DERIVE_KEY_TYPE_IS_SE05X(context)`

Are we using SE05X as crypto subsystem?

`SSS_DIGEST_TYPE_IS_SE05X(context)`

Are we using SE05X as crypto subsystem?

SSS_KEY_STORE_TYPE_IS_SE05X(keyStore)

Are we using SE05X as crypto subsystem?

SSS_MAC_TYPE_IS_SE05X(context)

Are we using SE05X as crypto subsystem?

SSS_OBJECT_TYPE_IS_SE05X(pObject)

Are we using SE05X as crypto subsystem?

SSS_RNG_CONTEXT_TYPE_IS_SE05X(context)

Are we using SE05X as crypto subsystem?

SSS_SESSION_TYPE_IS_SE05X(session)

Are we using SE05X as crypto subsystem?

SSS_SUBSYSTEM_TYPE_IS_SE05X(subsystem)

Are we using SE05X as crypto subsystem?

SSS_SYMMETRIC_TYPE_IS_SE05X(context)

Are we using SE05X as crypto subsystem?

SSS_TUNNEL_CONTEXT_TYPE_IS_SE05X(context)

Are we using SE05X as crypto subsystem?

SSS_TUNNEL_TYPE_IS_SE05X(context)

Are we using SE05X as crypto subsystem?

Enums

enum sss_s05x_sesion_prop_au8_t

SE050 Properties that can be represented as an array

Values:

enumerator **kSSS_SE05x_SessionProp_CertUID**

enum sss_s05x_sesion_prop_u32_t

SE050 Properties that can be represented as 32bit numbers

Values:

enumerator **kSSS_SE05x_SessionProp_CertUIDLen**

struct _sss_se05x_object

#include <fsl_sss_se05x_types.h>

An object (secure / non-secure) within a Key Store.

Public Members

uint32_t cipherType

cipherType type from [sss_cipher_type_t](#)

[SE05x_ECCurve_t](#) curve_id

If this is an ECC Key, the Curve ID of the key

uint8_t isPersistant

Whether this is a persistant 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_session

#include <fsl_sss_se05x_types.h>

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_tunnel_context

#include <fsl_sss_se05x_types.h>

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 terminates to

struct [SE05x_Applet_Feature_Disable_t](#)

#include <fsl_sss_se05x_types.h>

Used to disable Applet Features via [sss_se05x_set_feature](#)

Public Members

uint8_t [EXTCFG_FORBID_AES_GCM](#)

Disable feature AES_GCM B8b8

uint8_t [EXTCFG_FORBID_AES_GCM_EXT_IV](#)

Disable feature AES_GCM_EXT_IV B8b7

uint8_t [EXTCFG_FORBID_ECDH](#)

Disable feature ECDH B2b8

uint8_t [EXTCFG_FORBID_HKDF_EXTRACT](#)

Disable feature HKDF_EXTRACT B10b7

uint8_t [EXTCFG_FORBID_RSA_LT_2K](#)

Disable feature RSA_LT_2K B6b8

```
uint8_t EXTCFG_FORBID_RSA_SHA1
    Disable feature RSA_SHA1 B6b7
struct SE05x_Applet_Feature_t
    #include <fsl_sss_se05x_types.h>
    Used to enable 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_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 sss_se05x_aead_t
    #include <fsl_sss_se05x_types.h>
    Authenticated Encryption with Additional Data.

Public Members
sss_algorithm_t algorithm
```

Algorithm to be used

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

High level operation (encrypt/decrypt)

sss_se05x_session_t *session

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

struct **sss_se05x_asymmetric_t**

#include <fsl_sss_se05x_types.h>

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_derive_key_t**

#include <fsl_sss_se05x_types.h>

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**

```
#include <fsl_sss_se05x_types.h>
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](#)

```
#include <fsl_sss_se05x_types.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

[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](#)

```
#include <fsl_sss_se05x_types.h>
```

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_rng_context_t](#)

```
#include <fsl_sss_se05x_types.h>
```

Random number generator context

Public Members

`sss_se05x_session_t *session`

Pointer to the session

`struct sss_se05x_symmetric_t`

`#include <fsl_sss_se05x_types.h>`

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 can 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 its 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.

3.3.10.2 SSS SE05x Session types and APIs

`group sss_se05x_session`

Manage session.

Functions

`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.

`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 application has to call `sss_session_create` before calling `sss_session_open`.

Parameters

- **session** – [inout] Pointer to session context
- **subsystem** – [in] See [sss_session_open](#)
- **application_id** – [in] See [sss_session_open](#)
- **connection_type** – [in] See [sss_session_open](#)
- **connectionData** – [in] See [sss_session_open](#)

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.

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.

Note: [sss_session_open\(\)](#) must not be called concurrently from multiple threads. The application must ensure this.

Parameters

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

Returns

status

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.

Parameters

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

Returns

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](#)

Parameters

- **session** – [in] Session context
- **property** – [in] Value that is part of [sss_session_prop_u32_t](#)
- **pValue** – [out]

Returns

3.3.10.3 SSS SE05x Keystore types and APIs

groupsss_se05x_keystore

Manage session.

Functions

[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, a 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

- **keyStore** – [out] 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_se05x_key_store_context_free\(sss_se05x_key_store_t *keyStore\)](#)

Destructor for the key store context.

[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

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

[sss_status_t sss_se05x_key_store_erase_key\(sss_se05x_key_store_t *keyStore, sss_se05x_object_t *keyObject\)](#)

Delete / destroy allocated keyObject .

Parameters

- **keyStore** – The key store
- **keyObject** – The key object to be deleted

Returns

The sss status.

[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.

`sss_status_t sss_se05x_key_store_freeze_key(sss_se05x_key_store_t *keyStore, sss_se05x_object_t *keyObject)`

Not available for SE05X

`sss_status_t sss_se05x_key_store_generate_key(sss_se05x_key_store_t *keyStore, sss_se05x_object_t *keyObject, size_t keyBitLen, void *options)`

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

`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)

`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.

`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.

`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

If `keyObject == NULL`, then subsequent key injection does not use any KEK.

Parameters

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

Returns

The sss status.

Returns

The sss status.

`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.

`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.

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

Returns

3.3.10.4 SSS SE05x KeyObject types and APIs

groupsss_se05x_keyobj

Manage session.

Functions

`sss_status_t sss_se05x_key_object_allocate_handle(sss_se05x_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.

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.

Parameters

- **keyObject** – **[inout]** 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 = Persistant Key (Default) or Transient Key. See [sss_key_object_mode_t](#)

Returns

Status of object allocation.

`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.

`sss_status_t sss_se05x_key_object_get_access(sss_se05x_object_t *keyObject, uint32_t *access)`

Not Available for SE05X

`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 @ref sss_key_object_allocate_handle.%CRLF

SP% After calling this API, Ideally keyObject should become equivlant to as set after the calling of @ref
sss_key_object_allocate_handle api.

On SE05X, this API uses [Se05x_API_ReadType](#) and fetches parameters of the API.

Parameters

- **keyObject** – The key object
- **keyId** – **[in]** The key identifier

Returns

The sss status.

`sss_status_t sss_se05x_key_object_get_purpose(sss_se05x_object_t *keyObject, sss_mode_t *purpose)`

Not Available for SE05X

`sss_status_t sss_se05x_key_object_get_user(sss_se05x_object_t *keyObject, uint32_t *user)`

Not Available for SE05X

`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.

Parameters

- `keyObject` –
- `keyStore` –

Return values

- `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.

Returns

Status of the operation

`sss_status_t sss_se05x_key_object_set_access(sss_se05x_object_t *keyObject, uint32_t access, uint32_t options)`

Not Available for SE05X

`sss_status_t sss_se05x_key_object_set_eccgfp_group(sss_se05x_object_t *keyObject, sss_eccgfp_group_t *group)`

Not Available for SE05X

`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.

`sss_status_t sss_se05x_key_object_set_user(sss_se05x_object_t *keyObject, uint32_t user, uint32_t options)`

Not Available for SE05X

3.3.10.5 SSS SE05x Symmetric types and APIs

group`sss_se05x_symm`

Manage session.

Functions

`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.

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.
- **initialCounter** – **[inout]** Input counter (Always 16 bytes) (updates on return). When using internal IV algorithms (only encrypt) for SE051, initial Counter buffer will be filled with generated Initial counter.
- **lastEncryptedCounter** – **[out]** Output cipher of last counter, for chained CTR calls. NULL can be passed if chained calls are not used.
- **szLeft** – **[out]** Output number of bytes in left unused in lastEncryptedCounter block. NULL can be passed if chained calls are not used.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

```
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.

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.
- **destLen** – **[inout]** Length of output data in bytes. Buffer length on entry, reflects actual output size on return.

Return values

- **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.

Returns

Status of the operation

```
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.

Parameters

- **context** – Pointer to symmetric crypto context.
- **iv** – Buffer containing the symmetric operation Initialization Vector. When using internal IV algorithms (only encrypt) for SE051, iv buffer will be filled with generated Initialization Vector.
- **ivLen** – Length of the Initialization Vector in bytes.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

`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.

Parameters

- **context** – Pointer to symmetric crypto context.
- **iv** – Buffer containing the symmetric operation Initialization Vector. When using internal IV algorithms (only encrypt) for SE051, iv buffer will be filled with generated Initialization Vector.
- **ivLen** – Length of the Initialization Vector in bytes.
- **srcData** – Buffer containing the input data (block aligned).
- **destData** – Buffer containing the output data.
- **dataLen** – Size of input and output data buffer in bytes.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

`sss_status_t sss_se05x_cipher_one_go_v2(sss_se05x_symmetric_t *context, uint8_t *iv, size_t ivLen, const uint8_t *srcData, const size_t srcLen, uint8_t *destData, size_t *pDataLen)`

Symmetric cipher in one blocking function call. The function blocks current thread until the operation completes or an error occurs.

Parameters

- **context** – Pointer to symmetric crypto context.
- **iv** – Buffer containing the symmetric operation Initialization Vector. When using internal IV algorithms (only encrypt) for SE051, iv buffer will be filled with generated Initialization Vector.
- **ivLen** – Length of the Initialization Vector in bytes.
- **srcData** – Buffer containing the input data (block aligned).
- **srcLen** – Length of buffer srcData.
- **destData** – Buffer containing the output data.
- **dataLen** – Pointer to Size of buffer destData in bytes.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

`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\(\)](#).

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.
- **destLen** – **[inout]** Length of the output data in bytes. Buffer length on entry, reflects actual output size on return.

Return values

- **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.

Returns

Status of the operation

```
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.

```
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.

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 values

- **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.

Returns

Status of the operation

3.3.10.6 SSS SE05x Asymmetric types and APIs

groupsss_se05x_asym

Manage session.

Functions

```
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.

```
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.

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 values

- `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.

Returns

Status of the operation

```
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.

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 values

- `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.

Returns

Status of the operation

```
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.

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 values

- **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.

Returns

Status of the operation

`sss_status_t sss_se05x_asymmetric_sign(sss_se05x_asymmetric_t *context, const 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

`sss_status_t sss_se05x_asymmetric_sign_digest(sss_se05x_asymmetric_t *context, const uint8_t *digest, size_t digestLen, uint8_t *signature, size_t *signatureLen)`

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

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 values

- **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.

Returns

Status of the operation

`sss_status_t sss_se05x_asymmetric_verify(sss_se05x_asymmetric_t *context, const uint8_t *srcData, size_t srcLen, const uint8_t *signature, size_t signatureLen)`

Similar to [sss_se05x_asymmetric_verify_digest](#), but hashing/digest done by SE

`sss_status_t sss_se05x_asymmetric_verify_digest(sss_se05x_asymmetric_t *context, const uint8_t *digest, size_t digestLen, const uint8_t *signature, size_t signatureLen)`

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

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 values

- **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.

Returns

Status of the operation

3.3.10.7 SSS SE05x RNG types and APIs

groupsss_se05x_rng

Manage session.

Functions

`sss_status_t sss_se05x_rng_context_free(sss_se05x_rng_context_t *context)`

free random generator context.

Parameters

context – generator context.

Returns

status

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

Parameters

- **session** – Session context.
- **context** – random generator context.

Returns

status

`sss_status_t sss_se05x_rng_get_random(sss_se05x_rng_context_t *context, uint8_t *random_data, size_t dataLen)`

Generate random number.

Parameters

- **context** – random generator context.
- **random_data** – buffer to hold random data.
- **dataLen** – required random number length

Returns

status

3.3.10.8 SSS SE05x Digest types and APIs

groupsss_se05x_md

Manage session.

Functions`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.`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.

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 values

- **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.

Returns

Status of the operation

`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.

Parameters

- **context** – Pointer to digest context.
- **digest** – Output message digest
- **digestLen** – Message digest byte length

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

`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.

Parameters

`context` – Pointer to digest context.

Return values

- `kStatus_SSS_Success` – The operation has completed successfully.
- `kStatus_SSS_Fail` – The operation has failed.

Returns

Status of the operation

`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.

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 values

- `kStatus_SSS_Success` – The operation has completed successfully.
- `kStatus_SSS_Fail` – The operation has failed.

Returns

Status of the operation

`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.

Parameters

- `context` – Pointer to digest context.
- `message` – Buffer with a message chunk.
- `messageLen` – Length of the input buffer in bytes.

Return values

- `kStatus_SSS_Success` – The operation has completed successfully.
- `kStatus_SSS_Fail` – The operation has failed.

Returns

Status of the operation

3.3.10.9 SSS SE05x MAC types and APIs

`groupsss_se05x_mac`

Manage session.

Functions

```
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.

```
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.

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 values

- `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.

Returns

Status of the operation

```
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.

Parameters

- `context` – Pointer to mac context.
- `mac` – Output message MAC
- `macLen` – Computed MAC byte length

Return values

- `kStatus_SSS_Success` – The operation has completed successfully.
- `kStatus_SSS_Fail` – The operation has failed.

Returns

Status of the operation

```
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.

Parameters

`context` – Pointer to mac context.

Return values

- `kStatus_SSS_Success` – The operation has completed successfully.
- `kStatus_SSS_Fail` – The operation has failed.

Returns

Status of the operation

```
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.

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 values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

```
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.

Parameters

- **context** – Pointer to mac context.
- **message** – Buffer with a message chunk.
- **messageLen** – Length of the input buffer in bytes.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

```
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

3.3.10.10 SSS SE05x Key derivation types and APIs

groupsss_se05x_keyderive

Manage session.

Functions

```
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.

```
sss_status_t sss_se05x_derive_key_context_init(sss_se05x_derive_key_t *context, sss_se05x_session_t *session, 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.

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 values

- **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.

Returns

Status of the operation

[sss_status_t sss_se05x_derive_key_dh\(sss_se05x_derive_key_t *context, sss_se05x_object_t *otherPartyKeyObject, sss_se05x_object_t *derivedKeyObject\)](#)

Asymmetric key derivation Diffie-Hellmann The function cryptographically derives a key from another key. For example Diffie-Hellmann.

Parameters

- **context** – Pointer to derive key context.
- **otherPartyKeyObject** – Public key of the other party in the Diffie-Hellmann algorithm
- **derivedKeyObject** – **[inout]** Reference to a derived key

Return values

- **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.

Returns

Status of the operation

[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.

Deprecated:

Please use [sss_derive_key_one_go](#) instead

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.
- **derivedKeyObject** – **[inout]** Reference to a derived key
- **deriveDataLen** – Requested length of output
- **hkdfOutput** – Output buffer containing key derivation output
- **hkdfOutputLen** – Output containing length of hkdfOutput

Return values

- **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.

Returns

Status of the operation

`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.

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.
- **derivedKeyObject** – **[inout]** Reference to a derived key
- **deriveDataLen** – **[in]** Expected length of derived key.

Return values

- **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.

Returns

Status of the operation

`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.

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.
- **info** – [in] Input data buffer, typically with some fixed info.
- **infoLen** – [in] Length of info buffer in bytes.
- **derivedKeyObject** – Reference to a derived key
- **deriveDataLen** – [in] The derive data length

Return values

- **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.

Returns

Status of the operation

3.3.10.11 SSS SE05x AEAD types and APIs

groupsss_se05x_aead

Manage session.

Functions

`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.

`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.

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 values

- **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.

Returns

Status of the operation

`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 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).

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.
- **destLen** – [inout] 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 values

- **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.

Returns

Status of the operation

`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.

Parameters

- **context** – Pointer to aead crypto context.
- **nonce** – The operation nonce or IV. When using internal IV algorithms (only encrypt) for SE051, iv buffer will be filled with generated Initialization Vector.
- **nonceLen** – The length of nonce in bytes. For AES-GCM it must be ≥ 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 values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

`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.

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. When using internal IV algorithms (only encrypt) for SE051, iv buffer will be filled with generated Initialization Vector.
- **nonceLen** – The length of nonce in bytes. For AES-GCM it must be ≥ 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 (Default value is 16 bytes). 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 values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

`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.

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.
- **destLen** – **[inout]** Length of the output data in bytes. Buffer length on entry, reflects actual output size on return.

Return values

- **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.

Returns

Status of the operation

`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.

Parameters

- **context** – Pointer to aead crypto context
- **aadData** – Input buffer containing the chunk of AAD
- **aadDataLen** – Length of the AAD data in bytes.

Return values

- **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.

Returns

Status of the operation

3.3.10.12 SSS SE05x Tunnel types and APIs

groupsss_se05x_tunnel

Manage session.

Functions

`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.

`void sss_se05x_tunnel_context_free(sss_se05x_tunnel_context_t *context)`

Destructor for the tunnelling service context.

Parameters

context – **[out]** Pointer to tunnel context.

`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

- **context** – **[out]** Pointer to tunnel context. Tunnel context is updated on function return.
- **session** – Pointer to session this tunnelling service belongs to.

3.3.10.13 SSS SE05x I2C Master types and APIs

groupse050_i2cm

I2C Master APIs in SE050 for secure sensor.

Enums

enum SE05x_I2CM_Baud_Rate_t

Configuration for I2CM

Values:

enumerator kSE05x_I2CM_Baud_Rate_100Khz

enumerator kSE05x_I2CM_Baud_Rate_400Khz

enum SE05x_I2CM_securityReq_t

Additional operation on data read by I2C

Values:

enumerator kSE05x_Security_None

enumerator kSE05x_Sign_Request

enumerator kSE05x_Sign_Enc_Request

enum SE05x_I2CM_status_t

Status of I2CM Transaction

Values:

enumerator kSE05x_I2CM_Success

enumerator kSE05x_I2CM_I2C_Nack_Fail

enumerator kSE05x_I2CM_I2C_Write_Error

enumerator kSE05x_I2CM_I2C_Read_Error

enumerator kSE05x_I2CM_I2C_Time_Out_Error

enumerator kSE05x_I2CM_Invalid_Tag

enumerator kSE05x_I2CM_Invalid_Length

enumerator kSE05x_I2CM_Invalid_Length_Encode

enumerator kSE05x_I2CM_I2C_Config

enum SE05x_I2CM_TAG_t

I2C Master micro operation.

Values:

enumerator kSE05x_TAG_I2CM_Config

enumerator kSE05x_TAG_I2CM_Write

enumerator kSE05x_TAG_I2CM_Read

enum SE05x_I2CM_TLV_type_t

Types of entries in an I2CM Transaction

Values:

enumerator kSE05x_I2CM_None

Do nothing

enumerator kSE05x_I2CM_Configure

Configure the address, baudrate

enumerator kSE05x_I2CM_Write

Write to I2C Slave

enumerator **kSE05x_I2CM_Read**

Read from I2C Slave

enumerator **kSE05x_I2CM_StructuralIssue**

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, sss_se05x_attst_comp_data_t *pattest_data, uint8_t *rspbuffer, size_t *rspbufferLen, uint8_t noOftags)

Se05x_i2c_master_attst_txn.

I2CM Read With Attestation

Parameters

- **sess** – [in] session identifier
- **keyObject** – [in] Keyobject which contains 4 byte attestaion KeyId
- **p** – [inout] Array of structure type capturing a sequence of i2c master cmd/rsp transactions.
- **random_attst** – [in] 16-byte freshness random
- **random_attstLen** – [in] length of freshness random
- **attst_algo** – [in] 1 byte attestationAlgo
- **pattest_data** – [in] Data Related to Attestation process
- **rspbuffer** – [out] The read response
- **rspbufferLen** – [out] Length of the response
- **noOftags** – [in] Amount of structures contained in p

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.

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

Se05x_i2c_master_txn.

I2CM Transaction

Parameters

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

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.

struct **_SE05x_I2CM_cmd**

#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

3.3.10.14 SSS SE05x Attestation types and APIs

groupse05x_attest**Functions**

```
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

struct **sss_se05x_attst_comp_data_t**

```
#include <fsl_sss_se05x_types.h>
```

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]

Uinque ID of SE050
size_t chipIdLen
 Length of the Unique ID
uint8_t cmd[100]
 capdu for attestation
size_t cmdLen
 capdu Length of attestation
uint8_t objSize[2]
 object size
size_t objSizeLen
 object size Length
uint8_t signature[512]
 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

```
#include <fsl_sss_se05x_types.h>
```

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

3.3.10.15 SSS SE05x Other types and APIs

groupse05x_other

Functions

sss_status_t sss_se05x_set_feature(sss_se05x_session_t *session, [SE05x_Applet_Feature_t](#) feature, [SE05x_Applet_Feature_Disable_t](#) disable_features)

Set features of the Applet.

See [Se05x_API_SetAppletFeatures](#)

3.4 SSS APIs: SE051 vs SE050

SSS API name	SE051	SE050
sss_session_open()	Available	Available

SSS API name	SE051	SE050
<u>sss_session_close()</u>	Available	Available
<u>sss_key_store_set_key()</u>	POLICY_OBJ_FORBID_DERIVED_OUTPUT can be set to a SymmKey object	Available
<u>sss_key_store_generate_key()</u>	Available	Available
<u>sss_key_store_get_key()</u>	Available	Available
<u>sss_key_store_open_key()</u>	Available	Available
<u>sss_key_store_erase_key()</u>	Available	Available
<u>sss_key_store_context_free()</u>	Available	Available
<u>sss_key_object_init()</u>	Available	Available
<u>sss_key_object_allocate_handle()</u>	Available	Available
<u>sss_key_object_get_handle()</u>	Available	Available
<u>sss_key_object_free()</u>	Available	Available
<u>sss_symmetric_context_init()</u>	Available	Available
<u>sss_cipher_one_go()</u>	Available	Available
<u>sss_cipher_init()</u>	Available	Available
<u>sss_cipher_update()</u>	Available	Available
<u>sss_cipher_finish()</u>	Available	Available
<u>sss_cipher_crypt_ctr()</u>	Available	Available
<u>sss_symmetric_context_free()</u>	Available	Available
<u>sss_aead_context_init()</u>	Available	NA
<u>sss_aead_one_go()</u>	Available	NA
<u>sss_aead_init()</u>	Available	NA
<u>sss_aead_update_aad()</u>	Available	NA
<u>sss_aead_update()</u>	Available	NA
<u>sss_aead_finish()</u>	Available	NA
<u>sss_aead_context_free()</u>	Available	NA
<u>sss_digest_context_init()</u>	Available	Available
<u>sss_digest_one_go()</u>	Available	Available
<u>sss_digest_init()</u>	Available	Available
<u>sss_digest_update()</u>	Available	Available
<u>sss_digest_finish()</u>	Available	Available

SSS API name	SE051	SE050
<u>sss_digest_context_free()</u>	Available	Available
<u>sss_mac_context_init()</u>	Available	Available
<u>sss_mac_one_go()</u>	Available	Available
<u>sss_mac_init()</u>	Available	Available
<u>sss_mac_update()</u>	Available	Available
<u>sss_mac_finish()</u>	Available	Available
<u>sss_mac_context_free()</u>	Available	Available
<u>sss_asymmetric_context_init()</u>	Available	Available
<u>sss_asymmetric_encrypt()</u>	Available	Available
<u>sss_asymmetric_decrypt()</u>	Available	Available
<u>sss_asymmetric_sign_digest()</u>	Available	Available
<u>sss_asymmetric_verify_digest()</u>	Available	Available
<u>sss_asymmetric_context_free()</u>	Available	Available
<u>sss_derive_key_context_init()</u>	Available	Available
<u>sss_derive_key_go()</u>	Deprecated	Deprecated
<u>sss_derive_key_one_go()</u>	Derived Key object can be stored in Secure Element	Available
<u>sss_derive_key_sobj_one_go()</u>	Derived Key object can be stored in Secure Element	Available
<u>sss_derive_key_dh()</u>	Derived Key object can be stored in Secure Element	Available
<u>sss_derive_key_context_free()</u>	Available	Available
<u>sss_rng_context_init()</u>	Available	Available
<u>sss_rng_get_random()</u>	Available	Available
<u>sss_rng_context_free()</u>	Available	Available

3.5 Parameter Check & Conventions

3.5.1 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.

3.5.1.1 PCSC/CCID Interface and 64 Byte packet

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

3.5.2 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.

3.5.3 APIs

groupparam_check

Parameter Checks.

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

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

Project: SecureIoTMW

Defines

`ENSURE_OR_BREAK(CONDITION)`

If condition fails, break.

Sample Usage:

```
int SomeAPI()
{
    ...%%CRLF
    SP%% do {
        status = Operation1();
        ENSURE_OR_BREAK(0 == status);%%CRLF
        SP%% status = Operation2();
        ENSURE_OR_BREAK(0 == status);%%CRLF
        SP%% ...%%CRLF
        SP%% } while(0);%%CRLF
        SP%% 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;
...%%CRLF
SP%% value = Operation1();
ENSURE_OR_QUIT_WITH_STATUS_ON_ERROR(0 == value, status, ERR_FAIL);%%CRLF
SP%% value = Operation2();
ENSURE_OR_QUIT_WITH_STATUS_ON_ERROR(0 == value, status, ERR_NOT_ENOUGH_SPACE);%
%CRLF
SP%% ...
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

```
{
...%%CRLF
SP%% status = Operation1();
ENSURE_OR_GO_CLEANUP(0 == status);%%CRLF
SP%% status = Operation2();
ENSURE_OR_GO_CLEANUP(0 == status);%%CRLF
SP%% ...

cleanup:
return status;
}
```

ENSURE_OR_GO_EXIT(CONDITION)

If condition fails, goto :exit label

```
{
...%%CRLF
SP%% status = Operation1();
ENSURE_OR_GO_EXIT(0 == status);%%CRLF
SP%% status = Operation2();
ENSURE_OR_GO_EXIT(0 == status);%%CRLF
SP%% ...

exit:
return status;
}
```

ENSURE_OR_RETURN(CONDITION)

If condition fails, return

```
void SomeAPI()
{
...%%CRLF
SP%% status = Operation1();
ENSURE_OR_RETURN(0 == status);%%CRLF
SP%% status = Operation2();
ENSURE_OR_RETURN(0 == status);%%CRLF
SP%% ...%%CRLF
SP%% 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()
{
    ...%%CRLF
    SP%% status = Operation1();
    ENSURE_OR_RETURN_ON_ERROR(0 == status, ERR_FAIL);%%CRLF
    SP%% status = Operation2();
    ENSURE_OR_RETURN_ON_ERROR(0 == status, ERR_NOT_ENOUGH_SPACE);%%CRLF
    SP%% ...%%CRLF
    SP%% 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)

Waring 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)

Waring print of the parameter strCONDITION

Warning:

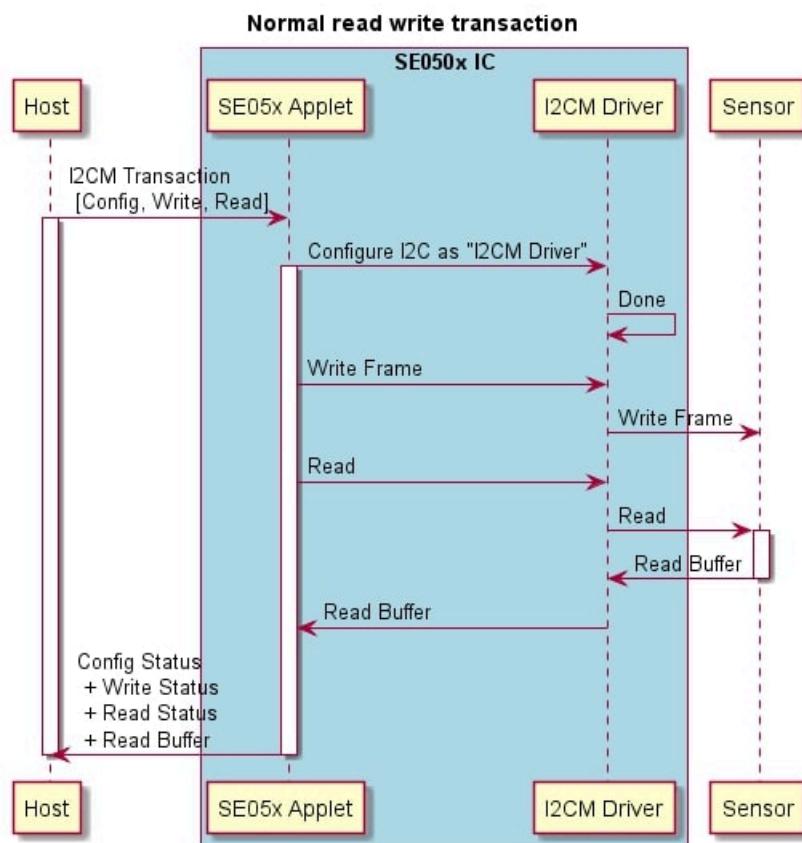
NX_ENSURE_MESSAGE is an internal message/API to this file. Do not use directly.

3.6 I2CM / Secure Sensor

For an example regarding this, see [Section 5.7.7I2C Master Example](#)

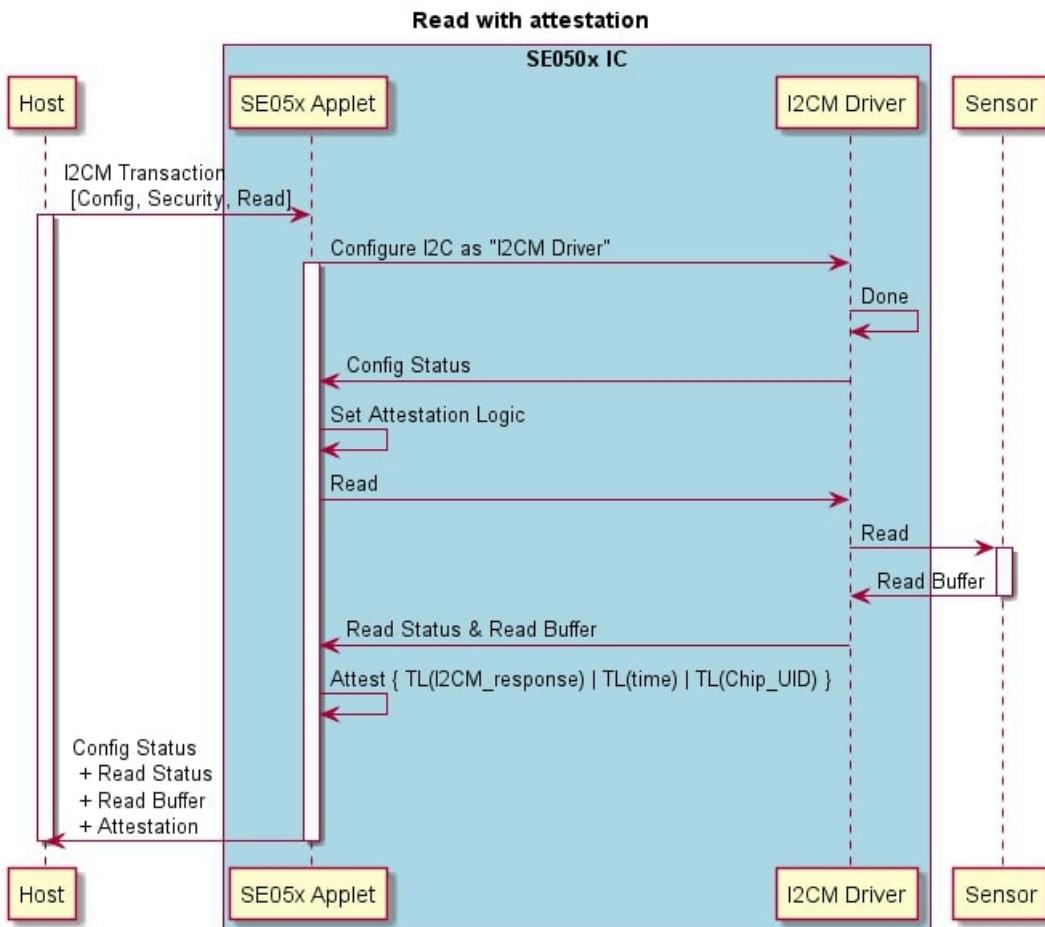
3.6.1 Normal Read/Write

The sequence to read from I2CM Sensor is as below:



3.6.2 Attested Read

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



3.6.3 Transaction

A sample I2CM Transaction can be performed as below:

Code:

3.6.4 Read with Attestation

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

Code:

3.6.5 I2C Master APIs

See [SSS SE05x I2C Master types and APIs](#)

3.7 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. Logging library `mwlog` is compiled and linked to all projects.

3.7.1 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**.

3.7.2 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.

3.7.3 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 a 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", rtStatus);
LOG_MAU8_D("FOO Command", rspbuf, rspbufLen);
```

3.7.3.1 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
```

3.7.3.2 Logging - Variable Names

Code:

```
uint32_t xu32val=0x12341234u;
uint8_t xu8val=0x44;
unsigned int some_int_value = 783;
unsigned char some_byte_value = 96; %%CRLF
SP%% 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);%%CRLF
SP%/* Logging that will be mis-interpreted */
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

```

3.7.3.3 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

```

3.7.3.4 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
```

3.7.4 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/log`. These files are machine generated and hence is not recommended to hand edit them.

3.7.5 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.

3.7.5.1 Full source-code

`hostlib/hostLib/libCommon/log/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 shoudl always be kept enabled */
#define NX_LOG_ENABLE_DEFAULT_ERROR 1%%CRLF
CRLF%/* Release - retail build */
#ifndef 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
```

3.7.5.2 Logging at component level

For example, changing logging level of App, hostlib/hostLib/libCommon/log/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
```

3.7.5.3 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  
#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](#).

3.8 Feature File - fsl_sss_ftr.h

The Plug & Trust Middleware uses a feature file to select/detect used/enabled features Within the middleware stack. When using CMake this file is automatically generated into the generated and used build directory. when not using CMake (e.g. using demo/example from the MCUXpresso MCU-SDK package, this file is kept at the root of the source folder.

3.8.1 When Using CMake

Please be careful that when you're using C Make this file is overwritten every time CMake is invoked or it re-generates the make files.

You do not have to hand modify fsl_sss_ftr.h feature file. Selections from CMake edit cache would automatically make relevant updates into the generated feature file.

This file is auto generated from simw-top\sss\inc\fsl_sss_ftr.h.in

3.8.2 When Using MCUXpresso IDE

As mentioned in above sections this file is kept in the root folder of the imported project. The file is filled with checks and balances so that at compile time some of the invalid selections are handled up front

3.8.3 fsl_sss_ftr_default.h

There is also provision to use a default fall-back file in case this feature file is not generated.

In the relevant parts the Middleware uses the following snippet to select the main or fall-back feature file:

```
#if defined(SSS_USE_FTR_FILE)  
#include "fsl_sss_ftr.h"  
#else  
#include "fsl_sss_ftr_default.h"  
#endif
```

It must be obvious that if the macro SSS_USE_FTR_FILE is not defined by the build system, default feature file gets used. In the reference demos and use cases from the Middleware, SSS_USE_FTR_FILE is always defined and only fsl_sss_ftr.h is used.

3.8.4 Using feature file to reduce code size

By setting below items to 0, either in CMake or the `fsl_sss_ftr.h` relevant sections of the code is removed from compilation and thereby reducing the code consumption.

3.8.5 SSS_HAVE_APPLET_SE05X_A

When we set to 1 can compile with this applet support(SE050 Type A (ECC))

3.8.6 SSS_HAVE_APPLET_SE05X_B

When we set to 1 can compile with SE05X_B applet support(SE050 Type A (RSA))

3.8.7 SSS_HAVE_APPLET_SE05X_C

When we set to 0 cannot compile with this applet support (SE050 (Super set of A + B))

Enable at-least one of 'PTMW_Applet' Not more than 1

3.8.7.1 SSSFTR_SW_TESTCOUNTERPART

For some of the demos we use cryptography both from the secure element and the host. For example, this makes an easy check for comparison where use comparison where we ask the host crypto to encrypt something and the secure element to do the counterpart (in this case encrypt) decrypt. Similarly for sign, verify.

Setting this to Zero, removes the implementation of counterpart.

3.8.7.2 SSSFTR_SW_ECC

When we set to 1, this feature exposes the asymmetric cryptography from the host for ECC. When using ECKey Authentication (See [Auth Objects : ECKey](#)) this feature needs to be enabled. Please note that `SSSFTR_SW_AES` also needs to be enabled for ECKey Authentication

3.8.7.3 SSSFTR_SW_RSA

When we set to 0, RSA related implementation from the host SW is removed.

3.8.7.4 Symmetric cryptography on Host

If for some reason there is no cryptography used at all on the host side then these macros can also be set to 0 to remove relevant code from the host.

- `SSSFTR_SW_AES`
- `SSSFTR_SW_KEY_GET`
- `SSSFTR_SW_KEY_SET`

3.8.7.5 SSSFTR_SE05X_AuthECKey

When set to 0 the authentication using (See [Auth Objects : ECKey](#)) mode is disabled from the Host.

3.8.7.6 SSSFTR_SE05X_AuthSession

When set to 0 then only Platform SCP or default session can be used to talk to the secure element.

3.8.7.7 SSSFTR_SE05X_AES

When set to 0 then the symmetric cryptography related APIs from the Secure Element is removed from compilation.

3.8.7.8 SSSFTR_SE05X_CREATE_DELETE_CRYPTOOBJ

When we set to zero the host never creates new crypto objects or neither delete them. under such situation crypto objects should already be created once in the lifetime of the secure element explicitly.

3.8.7.9 SSSFTR_SE05X_ECC

Feature related to various elliptic curves is removed when we set to 0.

3.8.7.10 SSSFTR_SE05X_KEY_GET

When there is no use case to fetch a key from the secure element to the host via the SSS APIs, this can be set to 0.

3.8.7.11 SSSFTR_SE05X_KEY_SET

When there is no use case to inject a key into the secure element from the host this can be set to 0. note that the keys can still be provisioned remotely for authenticated sessions this just removes the code from the host Middleware.

3.8.7.12 SSSFTR_SE05X_RSA

Removes the code related to RSA features of the secure element from the Middleware when we set to 0.

3.8.7.13 SSS_HAVE_ECC

When we set to 1, this feature exposes the ECC asymmetric cryptography. When we set to 0 the feature will Disable.

3.8.7.14 SSS_HAVE_RSA

When we set to 1, this feature exposes the RSA asymmetric cryptography. When we set to 0 the feature will Disable.

3.8.7.15 SSS_HAVE_EC_ED

When we set to 0 Edwards curve is Disabled. If we set to 1 curve will be Enabled.

3.8.7.16 SSS_HAVE_EC_MONT

when we set to 0 Montgomery Curve is Disabled. If we set to 1 the curve will be Enabled.

3.8.7.17 SSS_HAVE_MIFARE_DESFIRE

When we set to 0 MIFARE DESFire is Disabled. If we set to 1 MIFARE_DESFIRE will be Enabled.

3.8.7.18 SSS_HAVE_PBKDF2

When we set to 0 PBKDF2 will be Disabled. If we set to 1 PBKDF2 will be Enabled.

3.8.7.19 SSS_HAVE_TLS_HANDSHAKE

When we set to 0 TLS handshake APIs will be Disabled. If we set to 1 TLS handshake APIs will be Enabled.

3.8.7.20 SSS_HAVE_IMPORT

when we set to 0 import and export keys are Disabled. If we set 1 import export key are Enabled.

3.8.7.21 SSS_PFSCP_ENABLE

See [Section 3.9.1.1Configuring for OEF specific Platform SCP keys](#)

3.9 Platform SCP03 Authentication

3.9.1 MW Configuration

3.9.1.1 Configuring for OEF specific Platform SCP keys

File `sss/inc/fsl_sss_ftr.h.in` allows to configure the Plug & Trust MW to use Platform SCP keys for a specific OEF. Macros `SSS_PFSCP_ENABLE_*` are added for this. These macros can be used to configure the MW for PlatformSCP03 keys based on an OEF configuration. By default these remain disabled and default configuration is made based on `PTMW_SE05X_Ver` CMake option. For version `03_XX`, we use `SE050_DEVKIT` configuration, else we use `SE050E_0001A921` configuration.

Note:

Platform SCP03 supports both AES-128 (16 byte) and AES-256 (32 byte) keys.

Enabling any one of these macros would override the default configuration and enable the Platform SCP keys for that OEF. Also, macro `SSS_PFSCP_ENABLE_OTHER` allows the user to configure the MW in case user-defined Platform SCP keys are provisioned on the secure element. In that case, correct keys need to be updated in `ex_sss_tp_scp03_keys.h` before running the MW.

3.9.2 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 consider the environment variable. Is uses instead the keyfile specified on the command line.*

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

For Linux

For Windows

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

- ENC is 35C256458958A34F6136155F8209D6CD
- MAC is AF177D5DBDF7C0D5C10A05B9F1607F78
- DEK is A1BC8438BF77935B361A4425FE79FA29

The format of a reference file is as below:

```
# This is a comment, empty lines and comment lines allowed.  
ENC 35C256458958A34F6136155F8209D6CD # Trailing comment  
MAC AF177D5DBDF7C0D5C10A05B9F1607F78 # Optional trailing comment  
DEK A1BC8438BF77935B361A4425FE79FA29 # 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>

3.9.2.1 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.

3.10 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 : ECKey](#) : Uses ECC Key for authentication

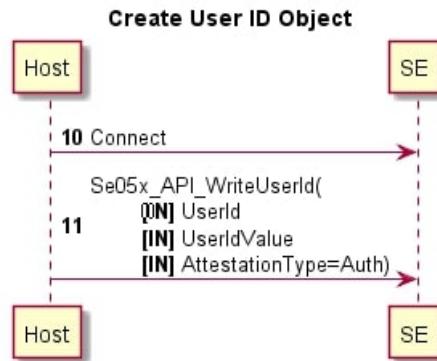
These are needed to have an authenticated session to the Applet. Also, in SE05X, access to many objects can be controlled via policies. Many of these polices are related to which Auth Object was used for the session to the SE05X.

3.11 Auth Objects : UserID

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

3.11.1 User ID - Provisioning / Injection

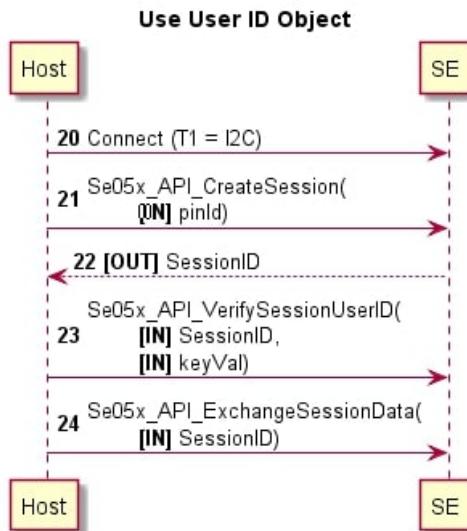
To provision / inject the key, the process is like this:

**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

3.11.2 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 Se05x_API_CreateSession() and use the 32bit id of UserId that we are going to use.
22	As a part of Se05x_API_CreateSession() API, Applet returns an 8 byte Session ID. We use this in future communication with the SE.

Table 2. Steps...continued

Step	Operation
23	Host calls Se05x_API_VerifySessionUserID() . 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 Se05x_API_ExchangeSessionData() API

3.11.3 User ID - Applet Spec Notes

From SE050 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

From SE051 APDU Spec:

3.3.1.9 UserID

A User ID object is a value which is used to logically group secure objects.

UserID objects can only be created as Authentication objects (see Section 3.3.3). They cannot be updated once created (i.e. the value of an existing UserID can not be changed).

A session that is opened by a UserID Authentication Object is not applying secure messaging (so no encrypted or MACed communication).

By default, the maximum number of allowed authentication attempts is set to infinite. Its length is 4 up to 16 bytes. It is intended for use cases where a trusted operating system on a host MCU/MPU is isolating applications based e.g. on application ID.

3.12 Auth Objects : AESKey

Applet SCP03 is applet level secure channel protocol, which involves 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.

3.13 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

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

3.13.2 ECKey - Use for connection / authentication

Authentication Phase:

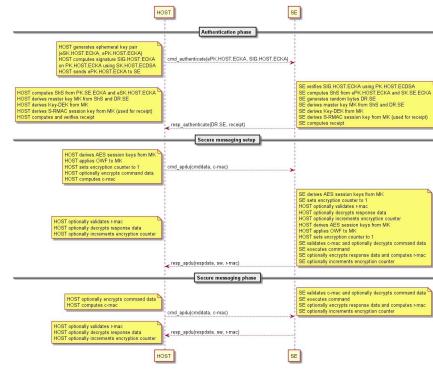
1. In the Secure Channel authentication phase, a Host-generated ephemeral key pair, the static SE key pair and SE-generated random data are used to compute a shared master secret
2. The Host generates an ephemeral ECC key pair and exchanges the public key component ePK.Host.ECKA with the SE. The ePK.Host.ECKA 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



3.14 Key Id Range and Purpose

These key ranges are used as a convention by the Middleware. Functionally (from applet point of view) they are the same. Only the “Applet reserved” keys have a special functionality as defined in the APDU specification of the secure element.

Purpose		Range
Customer keys	Start	0x00000001
	End	0x7FFFFFFF
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	0xFFFFFFFF
Required for EdgeLock 2GO	Start	0xF0000000
	End	0xFFFFFFFF

3.15 Authentication Keys

Authentication object type	KeyID
UserID	0x7DA00001
UserID	0x7DA00011
AESKey	0x7DA00002
AESKey	0x7DA00012
ECKey	0x7DA00003
ECKey	0x7DA00013

Note:

These authentication objects are used by the examples listed in [Section 5 — Demo and Examples](#) and are used as default authentication objects by the Middleware examples in case an applet session authentication is configured in the build options (See [Section 4.7.11PTMW_SE05X_Auth](#))

3.16 Trust provisioned KeyIDs

The secure element gets as well trust provisioned with further keys and certificates which are ready to be used. The below table lists the usually used ones. Not all keys and certificates are available on all types, for a complete definition refer to “AN12436 SE050 Configurations” respectively “AN12973 SE051 configurations (1.0)”

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

3.17 SCP03 with PUF

To keep Platform SCP03 keys secure, on the LPC55S e.g. PUF can be used. PUF will have the actual keys stored and we can perform cryptographic operations with it using HashCrypt block.

3.17.1 Activation Code

The Activation Code (AC) is a 1192-byte code used to start PUF. The AC is generated during `PUF_Enroll` operation. This must be generated once for the lifetime of the device and stored in PFR region of flash.

Each PUF has a different AC and cannot be used with any other device.

Note:

*For testing, we use pre-compiled activation code from `ex_scp03_puf.h` instead of reading from PFR. In actual use case, it **MUST** be stored and read from PFR.*

3.17.2 Key Code

For every key stored in PUF, we get a Key Code (KC) which is used to access the key. Hardware keys stored in PUF cannot be exported. SCP03 keys must be stored as hardware keys.

3.17.3 Using with LPC55S

PUF is integrated with `sss_session_open()` in the supplied LPC55S example. Use the following CMake configurations to compile with PUF on LPC55S:

- Host=lpcxpresso55s_s
- SCP=SCP03_SSS
- SE05X_Auth=PlatfSCP03

When we compile any application on LPC55S secure zone, it will try to read HW keys provisioned in PUF. If in case the keys are not provisioned in PUF, the implementation will fallback on software implementation.

Note:

You need to pass keyCodes in connectionData to `sss_session_open` instead of actual keys provisioned in PUF.

Only the static SCP03 keys are injected inside the PUF. Dynamic keys are derived from the static keys using CMAC operations with Hashcrypt module.

For example on how to enroll PUF and store SCP03 keys, refer [Key Injection to PUF](#).

3.18 SSS Heap Management

For effective heap management, macros `SSS_MALLOC`, `SSS_CALLOC` and `SSS_FREE` are available in `sss/port/ksdk/fsl_sss_types.h` for embedded build and in `sss/port/default/fsl_sss_types.h` for PC/Linux build.

```
#if defined(USE_THREADX_RTOS)

extern void *threadx_malloc(size_t size);
extern void *threadx_calloc(size_t size);
extern void threadx_free(void *ptr);

#ifndef SSS_MALLOC
#define SSS_MALLOC threadx_malloc
#endif // SSS_MALLOC

#ifndef SSS_FREE
#define SSS_FREE threadx_free
#endif // SSS_FREE
```

```
#ifndef SSS_CALLOC
#define SSS_CALLOC threadx_malloc
#endif // SSS_CALLOC

#elif (defined(USE_RTOS) && (USE_RTOS == 1))
#include "FreeRTOS.h"

void *pvPortCalloc(size_t num, size_t size); /*Calloc for Heap3/Heap4.*/

#ifndef SSS_MALLOC
#define SSS_MALLOC pvPortMalloc
#endif // SSS_MALLOC

#ifndef SSS_FREE
#define SSS_FREE vPortFree
#endif // SSS_FREE

#ifndef SSS_CALLOC
#define SSS_CALLOC pvPortCalloc
#endif // SSS_CALLOC

#else // !USE_RTOS & !USE_THREADX_RTOS

#include <stdlib.h>

#ifndef SSS_MALLOC
#define SSS_MALLOC malloc
#endif // SSS_MALLOC

#ifndef SSS_FREE
#define SSS_FREE free
#endif // SSS_FREE

#ifndef SSS_CALLOC
#define SSS_CALLOC calloc
#endif // SSS_CALLOC

#endif // USE_RTOS & USE_THREADX_RTOS
```

These macros are used for heap management operations in middleware and examples. All malloc/calloc/free calls should redirect to the same implementation. The same macro is also used for mbedTLS so that we are consistent across all malloc/free calls. In case of CMake configuration RTOS=FreeRTOS, we define the macros to use FreeRTOS implementation.

Warning:

Not using same implementation across the solution could lead to memory corruption.

It is recommended that these macros should be used for all applications. The user can also define their own implementation of heap APIs as platform dependent calls to malloc, calloc and free.

3.19 Secure Boot

In this section we describe the process of secure boot followed by Secondary Bootloader (SBL) to initialize SE05X and MCU for secure binding and communication.

3.19.1 Secure Boot flow

1. LPC55S ROM Bootloader verifies Secondary Bootloader (SBL) image (see AN12283) and hands over control to SBL
2. SBL performs Secure Binding and opens session to SE05X
3. SBL uses pre-provisioned key `K_PUB_OEM` in SE05X to verify secure application image
4. After successful verification, SBL loads verified image
5. (Optional) Secure Application can perform further initialization, non-secure image verification, hand over control to Non-Secure applications, or use SE050 with provided credentials (Secure Binding)

3.19.2 Secondary Bootloader

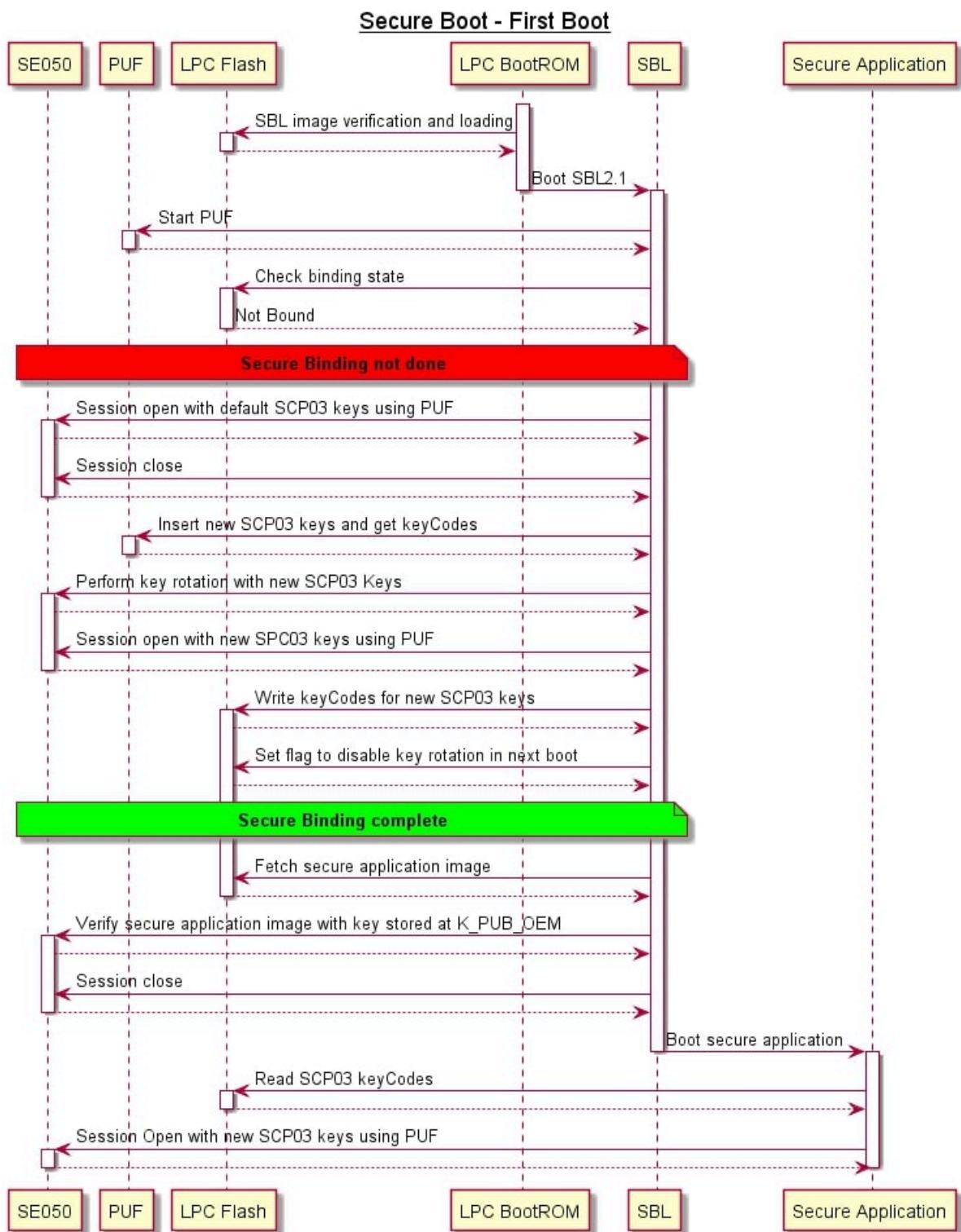
Secondary Bootloader (SBL) performs the following tasks to enable secure boot:

- **Secure Binding: Preparing SE05X for PlatformSCP**
 - Establish a pairing between the host MCU and the SE
 - MCU is only able to use services offered by the paired SE
 - SE is only able to provide services to the paired MCU
 - Mutually authenticated, encrypted SCP03 channel
- **Preparing host MCU for PlatformSCP session**
 - Securely storing PlatformSCP keys in PUF, protected by keyCodes
- **Verifying secure application**
 - Check the signature of secure application to verify that it is signed by a trusted key
- **KeyCode handover to secure application**
 - Handing over reference to ENC and MAC keyCodes prepared by SBL to secure application to open Platform SCP session

3.19.3 Secure Binding

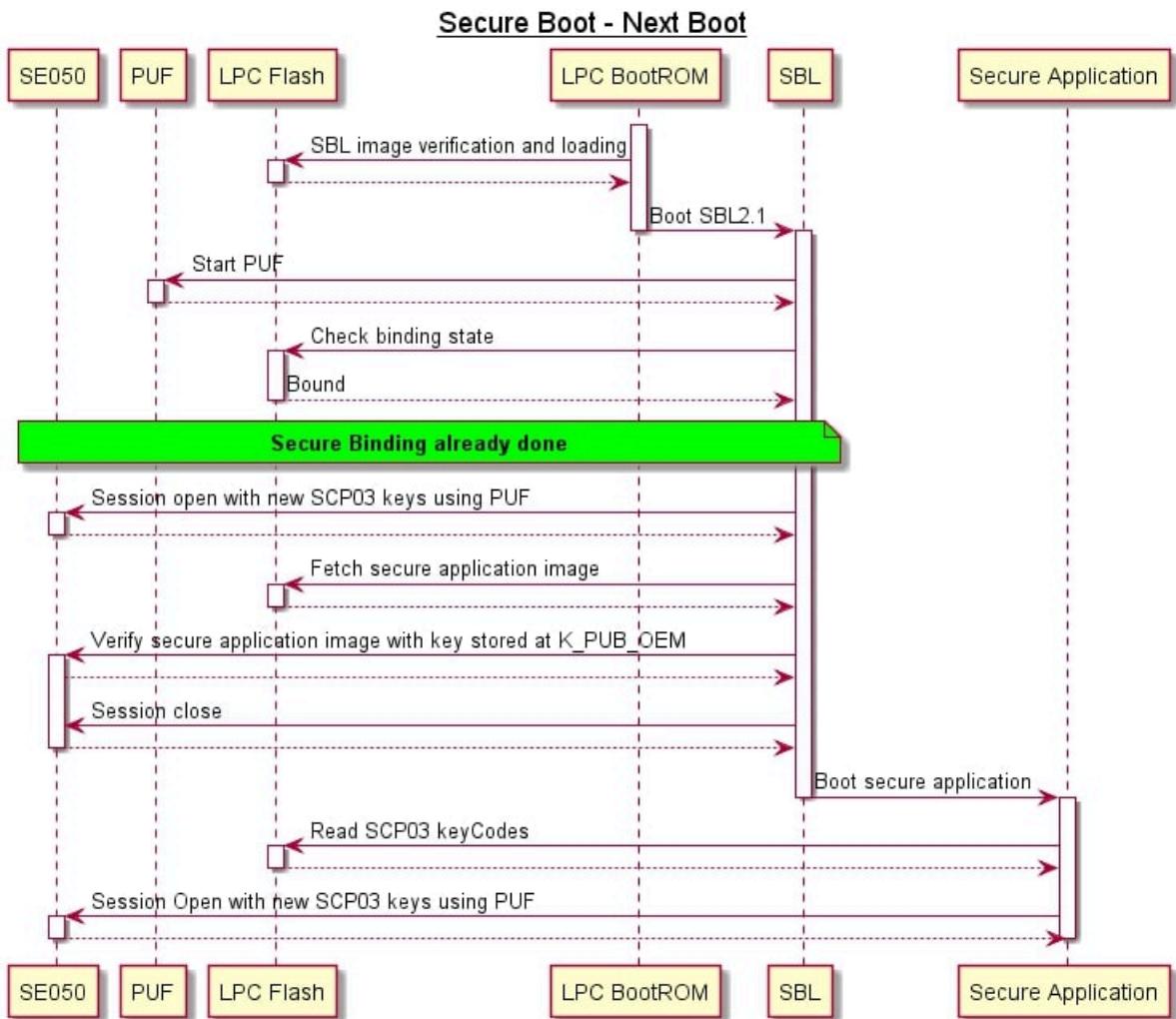
1. First boot

- SBL performs key rotation of the PlatformSCP key
- PlatformSCP keys are stored in PUF
- KeyCodes stored in flash to be used in subsequent boots
- Flag set in flash to disable key rotation in subsequent boots
- Verifies secure application



2. Next boot

- Checks for flag to disable key rotation
- Uses keyCodes from flash to verify secure application



Also see [Secure Boot Demo](#)

4 Building / Compiling

The Plug & Trust Middleware uses [CMake](#) for compiling / building.

Platform	Build Target
windows	Windows Build
Rpi	Raspberry Pi Build
imx	i.MX Linux Build
MCUs Build	Import MCUXpresso projects from SDK OR Freedom K64F (and Other MCUs) Build (CMake - Advanced)

4.1 Windows Build

Building the Plug & Trust MW on Windows enables to explore the MW stack and examples in a rich IDE. Only the low level communication to the secure element is done on a connected host as interface to the secure element. This process and the hardware setup to connect to SE05x is shown in AN12398 Quick start guide se05x Visual Studio Project examples.

4.1.1 Prerequisite

- Visual studio installed
- Python 3.8 or later 32 bit installed

Refer https://www.nxp.com/docs/en/application-note/AN12398-Quick_start_guide_se050_vs_projects.pdf for more details on prerequisite installation steps.

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
```

Note:

*There should not be any spaces in **SE05X_root_folder***

- 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

Note:

In case of using powershell, use env_setup.ps1 script to set environment variables.

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.

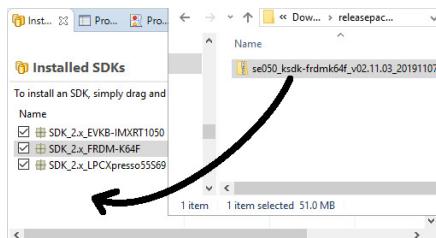
For more details refer - <https://www.nxp.com/docs/en/application-note/AN12396.pdf>

4.2.1 Prerequisite

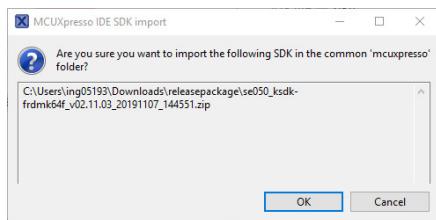
- MCUXpresso installed. Refer to [Setting up MCUXPresso IDE](#) for details on installation
- Plug & Trust Middleware SE050 MCUXPresso SDK Package is downloaded and available

4.2.2 Importing an example

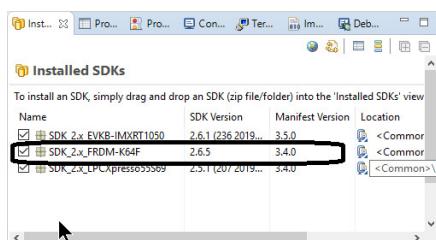
- Drag and drop SDK to “Install SDKs” windows of MCUXPresso. Even if you have older SDK, you can drag and drop. MCUXPresso will take the latest SDK based on version number.



- Accept the request to import the SDK.



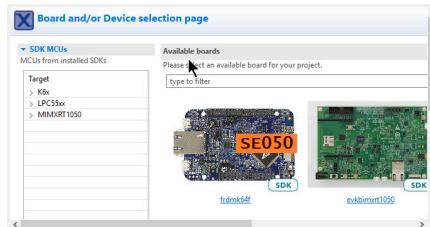
- You should see a new / updated entry. In this case version 2.6.5. Based on the release, this version may change. If you have Newer version of SDK, and you would like to use this version, you must delete the old SDK.



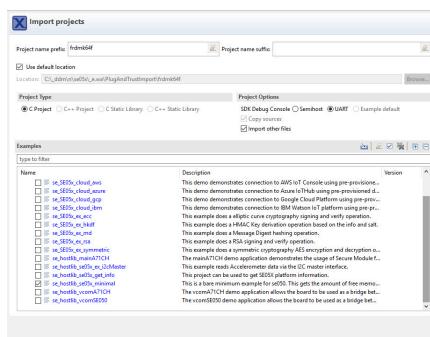
- From the “Quickstart Panel” → “Import SDK Example(s)”



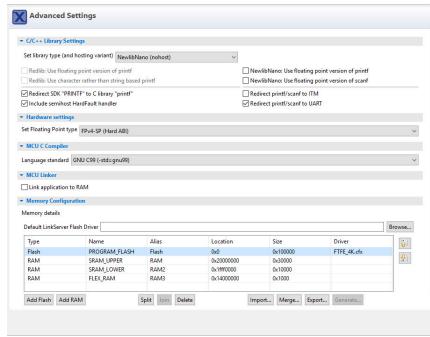
- You must see K64F with SE050 marked. For other SDKs, same convention follows. If you do not see SE050 there, it means you have newer version of the SDK and you must delete them.



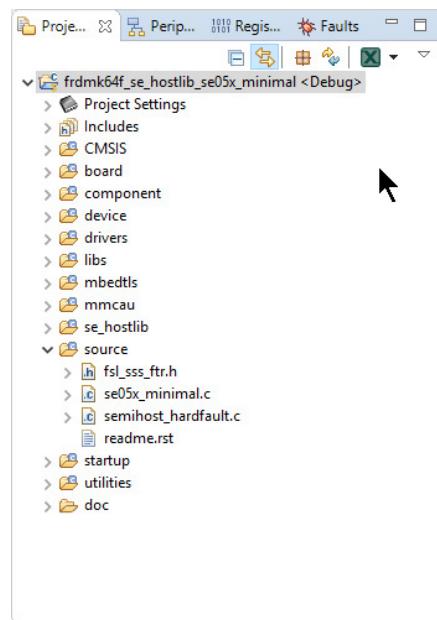
- When you import and click next, you will see support examples for that board.
 - Select the example you would like to run/test. In this case, we go for “minimal”



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

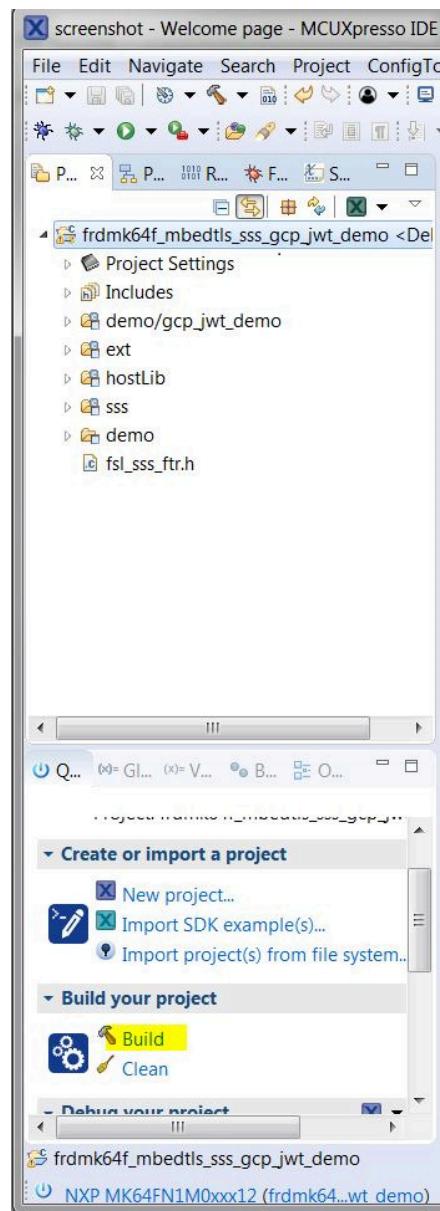


- Now the examples are ready to be tested.

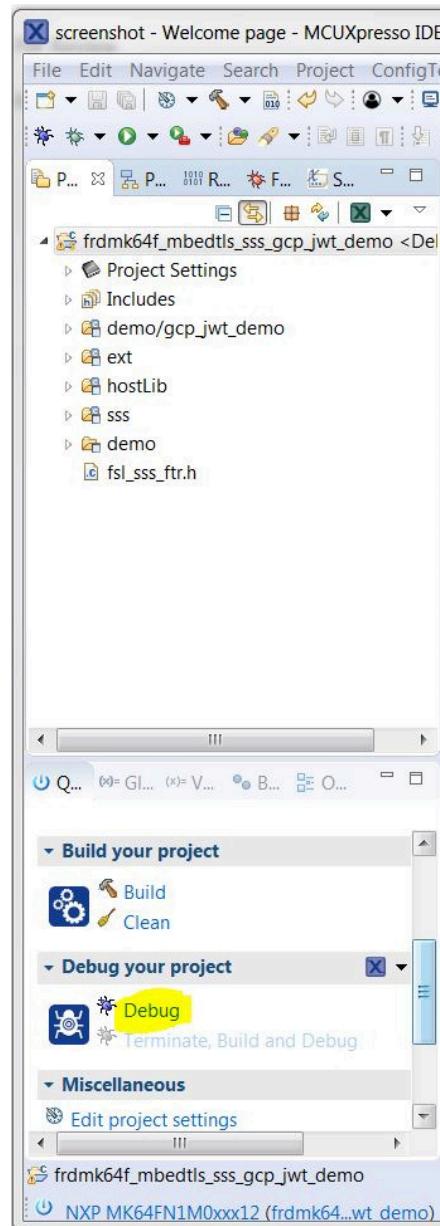


4.2.3 Running / Debugging example

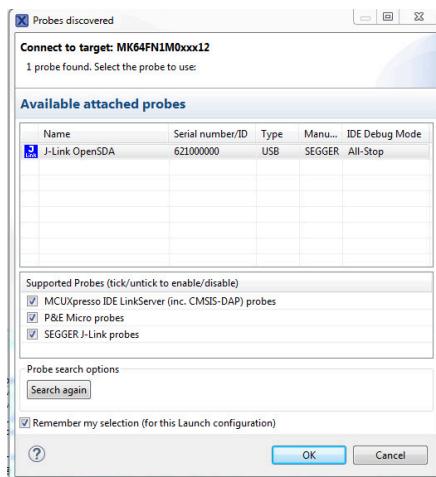
- Click on Build to compile and generate the executables



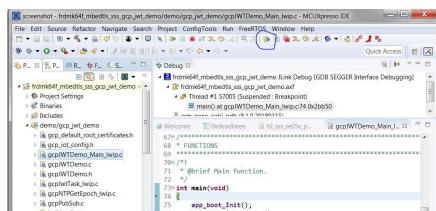
- Click on debug to program the binary into the device



- 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-adapter:OPENSDA>



- Click on the Resume button to start program execution



For UART, a serial terminal application(for e.g Tera Term) on PC for serial device needs to be configured as follows (in Tera Term goto Setup → Serial port)

- 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 (and Other MCUs) 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.

Note:

The steps are applicable for other supported MCUs supported also.

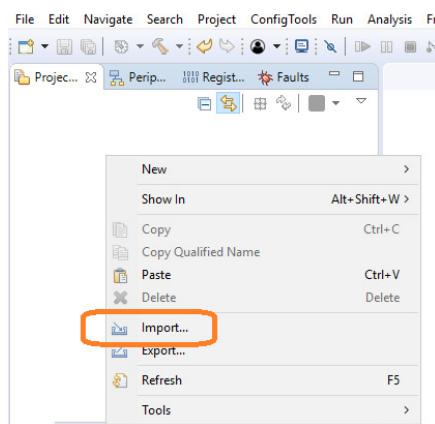
4.3.1 Prerequisite

- MCUXpresso installed and version correctly updated in `scripts/env_setup.bat` 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`

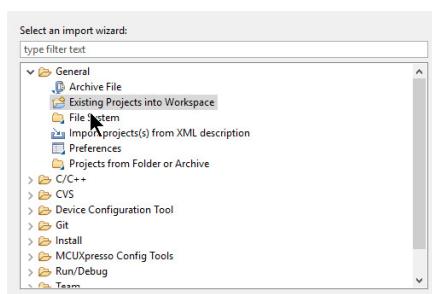
Refer <https://www.nxp.com/docs/en/application-note/AN12396.pdf> for more details on Prerequisite installation steps.

4.3.2 Importing the Project

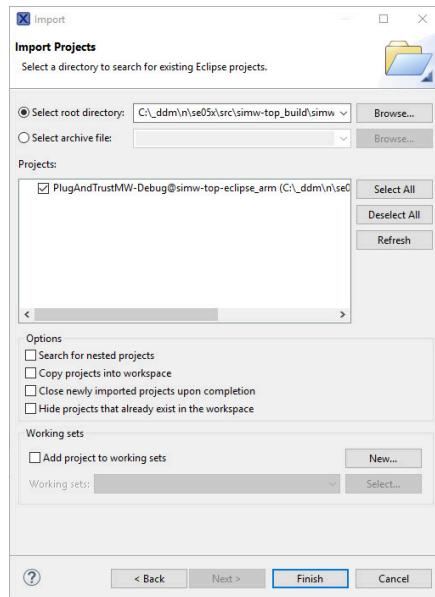
- Import a project using the option provided by MCUXpresso. Do not use the “Quickstart Panel” to import project.



- Import existing projects into Workspace

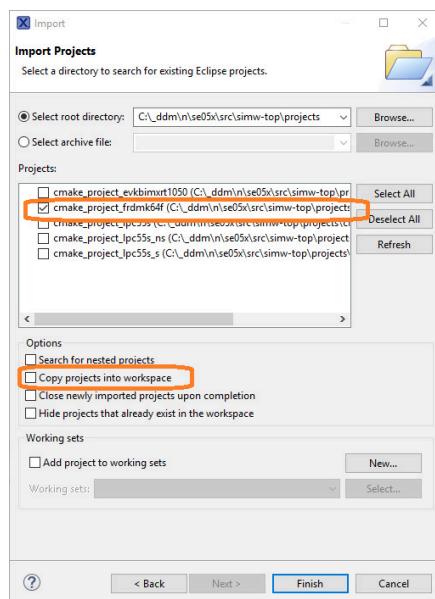


- First import the project generated by `scripts/create_cmake_projects.py`. We are going to use that work area to build the examples.
Never select “Copy projects into workspace” for this CMake configuration.

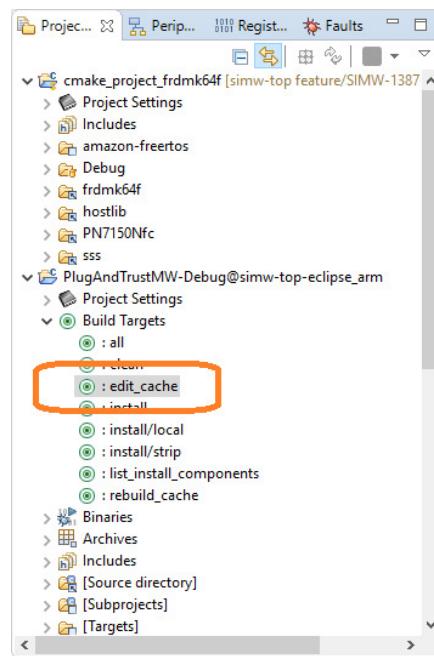


- Then import the project from “projects” directory. Import only the project for your board. In this case we import `cmake_project_XXXX`, where **XXXX** is the board name. This project is used to *debug* and *download* to the target. It’s a manually maintained project.

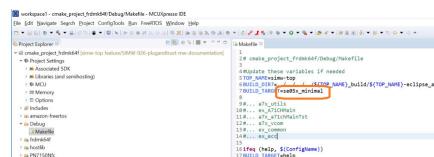
Never select “Copy projects into workspace” for this CMake configuration.



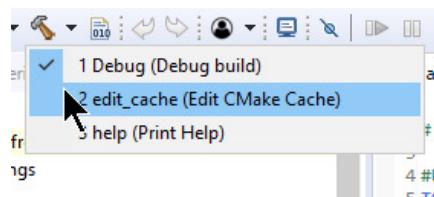
- You may have to run cmake's `edit_cache` to change the configuration of your example. e.g. choice of RTOS, Applet, Board, optimization level, etc.



- You would mostly have to change the target that you want to build and download. Modify `cmake_project_XXXX/Debug/Makefile`.
As you can see, the default selection is `se05x_Minimal`.
You can run the help target of this project a shown in next image to see which projects are supported.

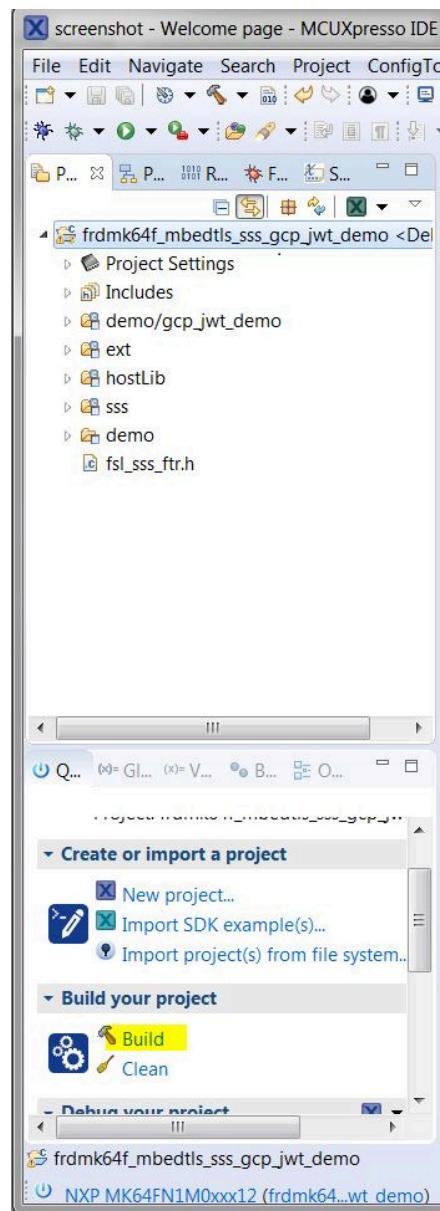


- `cmake_project_XXXX` project's settings also supports helpful target to `edit_cache` or list supported projects by that configuration.

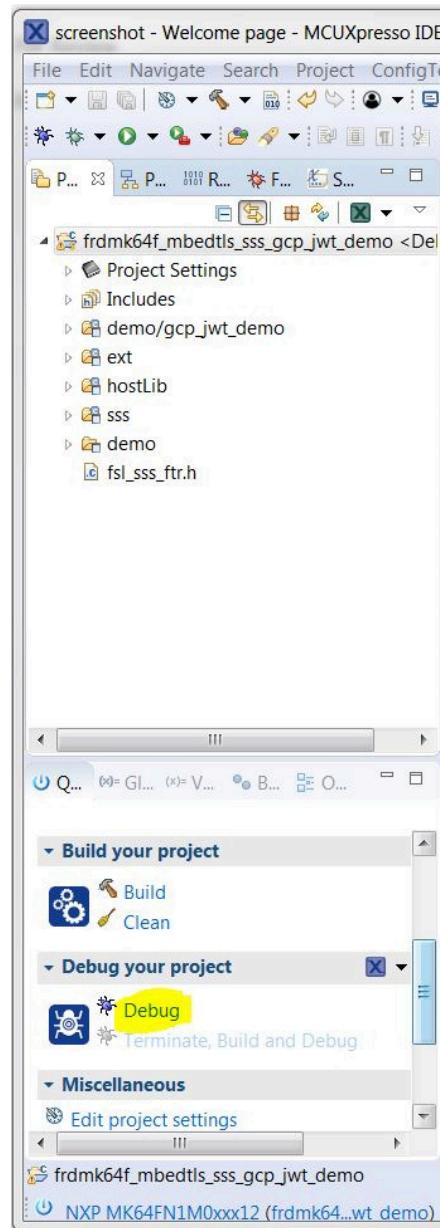


4.3.3 Running / Debugging example

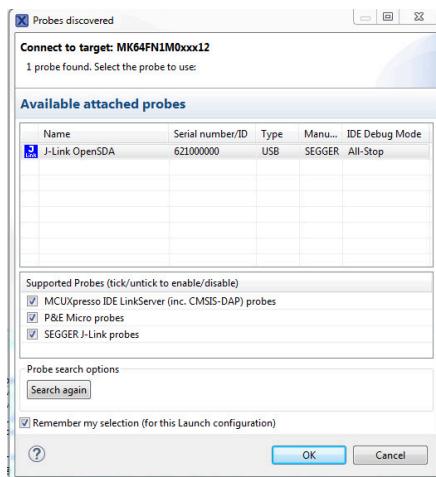
- Click on Build to compile and generate the executables



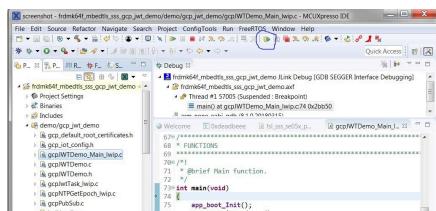
- Click on debug to program the binary into the device



- 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-adapter: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 (in Tera Term goto Setup → Serial port)

- 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 MCIMX8M-EVK development board. Refer to [Setup i.MX 8MQuad - MCIMX8M-EVK](#) for details on preparing the platform.

4.4.2 Build Instructions

- 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 # invoke this only once
cd .. /simw-top_build/imx_native_se050_t1oi2c
cmake --build .
make install
ldconfig /usr/local/lib
```

Default cmake options are shown below:

PTMW_Applet	SE050_C
PTMW_FIPS	None
PTMW_Host	iMXLinux
PTMW_HostCrypto	MBEDTLS
PTMW_Log	Default
PTMW_MBedTLS	2_X
PTMW_OpenSSL	3_0
PTMW_RTOS	Default
PTMW_SBL	None
PTMW SCP	None
PTMW_SE05X_Auth	None
PTMW_SE05X_Ver	07_02
PTMW_SMCOM	T1oI2C
WithCodeCoverage	OFF
WithNXPNFCRdLib	OFF
WithSSS_TestCounterPart	ON
WithSharedLIB	OFF
PTMW_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_t1oi2c
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_t1oi2c
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_t1oi2c
echo ${do # type double tab to enable command line completion hints

${doPTMW_Applet_SE05X_C_ON}           ${doPTMW_Host_evkmimxrt1060_ON}
${doPTMW_SE05X_Auth_AESKey_ON}         ${doPTMW_SE05X_Auth_None_ON}
${doPTMW_Applet_A71CL_ON}               ${doPTMW_Host_frdmk64f_ON}
...
...
${doPTMW_Host_Win10IoT_ON}             ${doPTMW_SCP_SCP03_SSS_ON}
```

```
cmake ${doPTMW_SE05X_Auth.UserID.ON} .
cmake --build .
```

Note:

The SW package also includes an install script (*scripts/imx/se05x_mw_install.sh*) that assists in installing the Plug&Trust MW on the iMX:

- Copy the Plug&Trust MW zip file into directory */home/root/install*
- Copy the install script *scripts/imx/se05x_mw_install.sh* into directory */home/root/install*
- Execute:

```
cd /home/root/install
chmod 755 se05x_mw_install.sh
./se05x_mw_install.sh <Plug&Trust>.zip
```

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_t1oi2c/bin* A subset of these examples is installed (upon executing *make install*) in */usr/local/bin*.

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

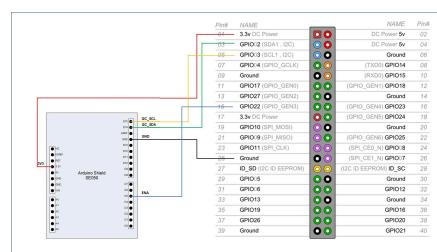
4.5 Raspberry Pi Build

Also Refer - <https://www.nxp.com/docs/en/application-note/AN12570.pdf>

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 Connecting SE05X with RaspberryPi



4.5.3 Enable Pin configuration for SE05X

Connect GPIO22 (P1_15) to enable pin of SE05X as indicated in the image above.

4.5.4 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 libsystemd-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_t1oi2c
cmake --build .
make install
ldconfig /usr/local/lib
```

Note:

If autodetection of RaspberryPi fails run the command with argument `python scripts/create_cmake_projects.py rpi`

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
NXPIInternal	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_t1oi2c
# 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 .`

4.5.5 SSS Examples

Above build steps will also build few sample examples for se05x test. Location: `simw-top_build/raspberrypi_native_se050_t1oi2c/bin`

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

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 `scripts\create_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 `scripts\env_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 PTMW_Applet

PTMW_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.

- DPTMW_Applet=None: Compiling without any Applet Support
- DPTMW_Applet=SE05X_A: SE050 Type A (ECC)
- DPTMW_Applet=SE05X_B: SE050 Type B (RSA)
- DPTMW_Applet=SE05X_C: SE050 (Super set of A + B), SE051, SE052
- DPTMW_Applet=SE051_H: SE051 with SPAKE Support
- DPTMW_Applet=AUTH: AUTH
- DPTMW_Applet=SE050_E: SE050E

4.7.2 PTMW_SE05X_Ver

PTMW_SE05X_Ver

Configures supported applet version. Set it to match the version used on the SE. Examples: for SE050A/B/C/F set 03_XX, for SE050E/51/A5000/SE052 set 07_02.

- DPTMW_SE05X_Ver=03_XX: SE050A/B/C/F
- DPTMW_SE05X_Ver=07_02: SE050E/51/A5000/SE052

4.7.3 PTMW_Host

PTMW_Host

Host where the software stack is running

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

- DPTMW_Host=Darwin: OS X / Macintosh
- DPTMW_Host=PCLinux32: PC/Laptop Linux with 32bit libraries
- DPTMW_Host=PCLinux64: PC/Laptop Linux with 64bit libraries
- DPTMW_Host=PCWindows: PC/Laptop Windows
- DPTMW_Host=Cygwin: Using Cygwin
- DPTMW_Host=frdmk64f: Embedded Kinetis Freedom K64F
- DPTMW_Host=evkmimxrt1060: Embedded Kinetis i.MX RT 1060
- DPTMW_Host=evkmimxrt1170: Embedded Kinetis i.MX RT1170
- DPTMW_Host=frdmmc9n947: Embedded Kinetis Freedom MCXN947
- DPTMW_Host=frdmmcxa153: Embedded Kinetis Freedom MCXA153
- DPTMW_Host=lpcxpresso55s: Embedded LPCXpresso55s (No demarcation of secure/non-secure world)
- DPTMW_Host=lpcxpresso55s_ns: Non Secure world of LPCXpresso55s

- DPTMW_Host=lpcxpresso55s_s: Secure world of LPCXpresso55s
- DPTMW_Host=iMXLinux: Embedded Linux on i.MX
- DPTMW_Host=Raspbian: Embedded Linux on Raspberry PI
- DPTMW_Host=Android: Android

4.7.4 PTMW_SMCOM

PTMW_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_V1_AM / JRCP_V2 / VCOM from PC.

- DPTMW_SMCOM=None: Not using any Communication layer
- DPTMW_SMCOM=JRCP_V2: Socket Interface New Implementation

-DPTMW_SMCOM=JRCP_V1: Socket Interface Old Implementation.

This is the interface used from Host PC when we run jrcpv1_server from the linux PC.

- DPTMW_SMCOM=JRCP_V1_AM: JRCP_V1 extended with Access manager features
- DPTMW_SMCOM=VCOM: Virtual COM Port
- DPTMW_SMCOM=T1oI2C: T=1 over I2C for SE050
- DPTMW_SMCOM=T1oI2C_GP1_0: GP Spec
- DPTMW_SMCOM=PCSC: CCID PC/SC reader interface

4.7.5 PTMW_HostCrypto

PTMW_HostCrypto

Counterpart Crypto on Host

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

- DPTMW_HostCrypto=MBEDTLS: Use mbedTLS as host crypto
- DPTMW_HostCrypto=OPENSSL: Use OpenSSL as host crypto

-DPTMW_HostCrypto=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.

-DPTMW_HostCrypto=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 PTMW_RTOS

PTMW_RTOS

Choice of Operating system

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

- DPTMW_RTOS=Default: No specific RTOS. Either bare metal on embedded system or native linux or Windows OS
- DPTMW_RTOS=FreeRTOS: Free RTOS for embedded systems

4.7.7 PTMW_mbedTLS_ALT

PTMW_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

- DPTMW_mbedTLS_ALT=SSS: Use SSS Layer ALT implementation
- DPTMW_mbedTLS_ALT=PSA: Enable TF-M based on PSA as ALT
- DPTMW_mbedTLS_ALT=None: Not using any mbedTLS_ALT

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

4.7.8 PTMW SCP

PTMW SCP

Secure Channel Protocol

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

- DPTMW_SCP=None
- DPTMW_SCP=SCP03_SSS: Use SSS Layer for SCP. Used for SE050 family.

4.7.9 PTMW_FIPS

PTMW_FIPS

Enable or disable FIPS

This selection mostly impacts tests, and generally not the actual Middleware

- DPTMW_FIPS=None: NO FIPS
- DPTMW_FIPS=SE050: SE050 IC FIPS

4.7.10 PTMW_SBL

PTMW_SBL

Enable/Disable SBL Bootable support

This option is to enable/disable boot from SBL by switching linker address

- DPTMW_SBL=None: Not SBL bootable
- DPTMW_SBL=SBL_LPC55S: SE050 based LPC55S SBL bootable

4.7.11 PTMW_SE05X_Auth

PTMW_SE05X_Auth

SE050 Authentication

This setting is used by examples to connect using various options to authenticate with the Applet. The SE05X_Auth options can be changed for MCU-SDK Demos and Examples. To change SE05X_Auth option follow below steps. Set flag SSS_HAVE_SCP_SCP03_SSS to 1 and Reset flag SSS_HAVE_SCP_NONE to 0. To change SE05X_Auth option other than None and PlatfSCP03, execute se05x_Delete_and_test_provision.exe in order to provision the Authentication Key. To change SE05X_Auth option to ECKey or ECKey_PlatformSCP03, Set additional flag SSS_HAVE_HOSTCRYPTO_ANY to 1.

- DPTMW_SE05X_Auth=None: Use the default session (i.e. session less) login
- DPTMW_SE05X_Auth=UserID: Do User Authentication with UserID
- DPTMW_SE05X_Auth=PlatfSCP03: Use Platform SCP for connection to SE

-DPTMW_SE05X_Auth=AESKey: Do User Authentication with AES Key

Earlier this was called AppletSCP03

-DPTMW_SE05X_Auth=ECKey: Do User Authentication with EC Key

Earlier this was called FastSCP

-DPTMW_SE05X_Auth=UserID_PlatformSCP03: UserID and PlatformSCP03

-DPTMW_SE05X_Auth=AESKey_PlatformSCP03: AESKey and PlatformSCP03

-DPTMW_SE05X_Auth=ECKey_PlatformSCP03: ECKey and PlatformSCP03

4.7.12 PTMW_Log

PTMW_Log

Logging

-DPTMW_Log=Default: Default Logging

-DPTMW_Log=Verbose: Very Verbose logging

-DPTMW_Log=Silent: Totally silent logging

4.7.13 CMAKE_BUILD_TYPE

CMAKE_BUILD_TYPE

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

For embedded builds, this choice 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

4.7.14 PTMW_OpenSSL

PTMW_OpenSSL

For PC, which OpenSSL to pick up

On Linux based builds, this option has no impact, because the build system picks up the default available/installed OpenSSL from the system directly.

-DPTMW_OpenSSL=1_1_1: Use 1.1.1 version (Only applicable on PC)

-DPTMW_OpenSSL=3_0: Use 3.0 version (Only applicable on PC)

4.7.15 PTMW_MBedTLS

PTMW_MBedTLS

Which MBedTLS version to choose

-DPTMW_MBedTLS=2_X: Use 2.X version

-DPTMW_MBedTLS=3_X: Use 3.X version

4.7.16 PTMW_SE_RESET_LOGIC

PTMW_SE_RESET_LOGIC

Reset logic of Secure Element. The value of this will be used for 'SE_RESET_LOGIC' define in 'se05x_ic_reset' function. Select 1 (Active High) for SE050 and 51. Select 0 (Active low) for SE052.

-DPTMW_SE_RESET_LOGIC=1: reset logic is set to 1

-DPTMW_SE_RESET_LOGIC=0: reset logic is set to 0

4.7.17 Feature Control

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

See [Section 3.8.4 Using feature file to reduce code size](#) for details on it's usage and relevance.

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_AuthECKey

SE05X Secure Element : Authenticate via ECKey

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_CRYPTOOBJ

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.

4.7.18 Deprecated Defines

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

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.

WithAccessMgr_UxSocket

- Compile Access Manager with UNIX socket support (Default is STREAM sockets).
- Default is OFF
- Compile Access Manager with unix socket support.

4.7.19 NXP Internal Options

These options are not supported outside NXP.

NXPIInternal

- 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

4.7.20 Other Variables

WithExtCustomerTPMCode

- Include code from/customer/tpm2
- Default is OFF
- Include code from external tpm2 folder. This way, TPM 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

5 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

MCU-SDK

Embedded platforms like FRDM K64F, i.MX RT1060, LPC55S, MCXN947, MCXA153

MCU-SDK-CLOUD

Embedded platforms like FRDM K64F, i.MX RT1060, LPC55S, that can connect to cloud.

LINUX

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

PC

Windows PC

ALL

MCU-SDK, LINUX, PC

5.1.2 SSS APIs Examples

Demo	Description	Platforms supported	SE supported	Authentication required
Section 5.2.1.1	ECC Example Inject ECC Key and use it for sign and verify operation	ALL	SE05X (A and C)	
Section 5.2.1.2	RSA Example Generate RSA key and use it for signin and verify operation	ALL	SE05X (B and C)	
Section 5.2.1.3	Symmetric AES Example Inject AES key, encrypt and decrypt data with it	ALL	SE05X	
Section 5.2.1.4	HKDF Example 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	

Demo	Description	Platforms supported	SE supported	Authentication required
Section 5.2.1.5	Message Digest Example Message Digest hashing operation. Calculate SHA256 over data.	ALL	SE05X	
Section 5.2.1.6	HMAC Example Inject HMAC key and calculate a HMAC	ALL	SE05X	
Section 5.2.1.7	ECDH Example Inject ECC key into SE and derive a key using ECDH from the SE into the host keystore.	ALL	SE05X	

5.1.3 Cloud connectivity Examples

Demo	Description	Platforms supported	SE supported	Authentication required
Section 5.3.1	AWS Demo for MCU-SDK Connect to Amazon Web Services IoT Core		SE05X	
Section 5.3.2	AWS Demo for iMX Linux / RaspberryPi Connect to Amazon Web Services	LINUX	SE05X	
Section 5.3.3	Azure Demo for MCU-SDK Connect to Microsoft Azure		SE05X	
Section 5.3.4	Azure Demo for iMX Linux / Raspberry Pi Connect to Microsoft Azure	LINUX	SE05X	
linux-demos-greengrass	linux-demos-greengrass Connect as AWS Greengrass Core	Raspberry PI	SE05X	

5.1.4 OpenSSL Engine Examples

Demo	Description	Platforms supported	SE supported	Authentication required
Section 5.4.1	OpenSSL Engine / Provider: TLS Client example for iMX/Rpi3 Setting up a TLS Link using OpenSSL Engine	LINUX	SE05X	

5.1.5 mbedTLS Examples

Demos regarding the mbedTLS ALT implementation. See [Introduction on MbedTLS ALT \(2.x\) Implementation](#)

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

5.1.6 OPC UA Examples

Demo	Description	Platforms supported	SE supported	Authentication required
Section 5.5.1	OPC UA (Open62541) Demo OPC UA Server	PC, iMX6	SE05X	

5.1.7 PSA / TF-M Examples

Demo	Description	Platforms supported	Authentication required
Section 5.6.1	PSA Non Secure Example PSA Secure Non-secure example	LPC55S	

5.1.8 SE05X Specific Examples

Demo	Description	Platforms supported	SE supported	Authentication required
Section 5.7.1	SE05X Minimal example Showcase usage of SE05X low level APIs	ALL	SE05X	
Section 5.7.2	SE05X Get Info example Showcase Platform details of SE05X	ALL	SE05X	
Section 5.7.3	APDU Player Demo Send RAW APDUs to SE050	PC, LINUX	SE05X	None / Platform SCP03
Section 5.7.4	Using policies for secure objects Showcase usage of policies	ALL	SE05X	
Section 5.7.5	Get Certificate from the SE Read the certificate from the SE and store it on the file system.	ALL (With mbedTLS Only)	SE05X	
Section 5.7.6	SE05X Rotate PlatformSCP Keys Demo Showcase Rotation of SE05X PlatformSCP03 Keys	ALL	SE05X	Platform SCP03
Section 5.7.7	I2C Master Example Showcase usage of I2CM interface of SE050	ALL	SE05X	
Section 5.7.8	SE05X WiFi KDF Example Showcase usage of PBKDF	ALL	SE05X	
Section 5.7.9	SE05X Export Transient objects Export transient objects	PC, LINUX	SE05X	
Section 5.7.10	SE05X Import Transient objects Import transient objects	PC, LINUX	SE05X	
Section 5.7.11	Import External Object Prepare Create ImportExternlObject raw APDU	PC, LINUX	SE05X	ECKey
Section 5.7.12	SE05X Mandate SCP example	ALL	SE05X	
Section 5.7.13	Read object with Attestation Demonstrate how to read object with attestation	ALL	SE05X	

Demo	Description	Platforms supported	SE supported	Authentication required
Section 5.7.14	SE05X Transport Lock example Show transport lock feature	PC, LINUX	SE05X	None / Platform SCP03
Section 5.7.15	SE05X Transport UnLock example Show transport unlock feature	PC, LINUX	SE05X	None / Platform SCP03
Section 5.7.16	SE05X Timestamp Demonstrate increment of timestamp inside SE	ALL	SE05X	
Section 5.7.17	SE05X PCR example Demonstrate PCR feature as Policy	ALL	SE05X	
Section 5.7.18	Configuring Applet Features Demonstrate how to configure applet features	PC, LINUX	SE05X	ECKey / ECKey-Platform SCP03 (FEATURE ID)
write-apdu-to-buf	write-apdu-to-buf Demonstrate how to write APDU to buffer	ALL	SE05X	
Section 5.7.19	Inject Certificate into SE Example to showcase injection of certificates into SE	ALL	SE05X	
Section 5.7.20	SE05X Read State example Example to Read the LockState, RestrictMode and PlatformSCPRequest of SE	ALL	SE051 / SE052	
Section 5.7.21	SE05X Personalization Remove RSA Key Generation Module Example to showcase to delete rsa key generation module from SE	ALL	SE051	
Section 5.7.22	DEMO for Personalization of SE051 Personalization of SE051	PC, LINUX	SE051	
Section 5.7.23	SE05X MultiThread demo Showcase opening multiple sessions using multiple threads	ALL (Except MSVC compiler)	SE05X	
Section 5.7.24	SE05X Invoke Garbage Collection Example Invoke Garbage Collection	ALL	SE05X	
Section 5.7.25	ECC Concurrent Example	ALL	SE05X	None/ UserId/ AESkey/ ECkey
Section 5.7.26	Symmetric Multi Step Concurrent Example	ALL	SE05X	None/ UserId/ AESkey/ ECkey
Section 5.7.31	Secure Authenticator (Qi) Authentication demo	ALL	SE05X	

5.1.9 Examples that use OpenSSL

Demo	Description	Platforms supported	SE supported	Authentication required
Section 5.8.1	Tool to create Reference key file Native example to generate refKeys. (Only for NIST-P256 curve).	LINUX, PC	SE05X	
Section 5.8.2	Building a self-signed certificate 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.

Number	Description
Section 5.11.1	Ease of Use configuration - Azure IoT Hub
Section 5.11.2	Ease of Use configuration - AWS IoT Console

5.1.11 SEMS Demos

See [PTMW_Applet](#). You need to compile with `-DSE05X_Ver=06_00 -DSE05X_Auth=None`

Demo	Description	Platforms supported	SE Supported
Section 5.12.1	SEMS Lite Agent Demo (sems_lite_ex_update)	ALL	SE051
Section 5.12.2	SEMS Lite CLI APP	PC, LINUX	SE051

5.1.12 LPC55S-PUF Based examples

Demo	Description	Platforms supported	SE supported	Authentication required
Section 5.13.1	Key Injection to PUF Example to demonstrate inject PlatformSCP keys into PUF	LPC55S	SE05X	
Section 5.13.2	Key Rotation using PUF Example to demonstrate PlatformSCP key rotation using PUF	LPC55S	SE05X	Platform SCP03
Section 5.13.3	Secure Boot Demo Example to demonstrate Secure Binding with LPC55S and SE05X using PUF	LPC55S	SE05X	Platform SCP03

5.1.13 EdgeLock 2GO Agent example

Demo	Description	Platforms supported	SE supported	Authentication required
Section 6.8	EdgeLock 2GO Agent Examples Example of usage of the EdgeLock 2GO Client	PC, iMX6, iMX8, FRDM K64F, LPC55S	SE05X	None / PlatformSCP03

5.2 SSS API Examples

5.2.1 ECC Example

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

Refer - simw-top/sss/ex/ecc/ex_sss_ecc.c

5.2.1.1 Prerequisites

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

5.2.1.2 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\(\)`](#)

5.2.1.3 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 !!!
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.

Refer - simw-top/sss/ex/rsa/ex_sss_rsa.c

5.2.2.1 Prerequisites

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

5.2.2.2 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_CRT`](#) 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.

5.2.2.3 Console output

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

5.2.3 Symmetric AES Example

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

Refer - simw-top/sss/ex/symmetric/ex_sss_symmetric.c

5.2.3.1 Prerequisites

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

5.2.3.2 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`](#)

5.2.3.3 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.

Refer - simw-top/sss/ex/hkdf/ex_sss_hkdf.c

5.2.4.1 Prerequisites

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

5.2.4.2 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\(\)](#)

5.2.4.3 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.

Refer - simw-top/sss/ex/md/ex_sss_md.c

5.2.5.1 Prerequisites

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

5.2.5.2 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_SHA256 from [sss_algorithm_t](#)
- kMode_SSS_Digest from [sss_mode_t](#)
- [sss_digest_one_go\(\)](#)

5.2.5.3 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.

Refer - simw-top/sss/ex/hmac/ex_sss_hmac.c

5.2.6.1 Prerequisites

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

5.2.6.2 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\(\)](#)

5.2.6.3 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.

Refer - simw-top/sss/ex/ecdh/ex_sss_ecdh.c

5.2.7.1 Prerequisites

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

5.2.7.2 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\(\)](#)

5.2.7.3 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
```

5.2.8 ECC NIST256 Key Attestation Example

This project demonstrates ecc nist256 key attestation and verification with another ecc nist256 key using SSS API

Refer - simw-top/sss/ex/attest_ecc/ex_sss_ecc_attest.c

5.2.8.1 Prerequisites

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

5.2.8.2 About the Example

This example reads a nist256 public key with attestation.

It uses the following APIs and data types:

- [sss_key_store_set_key\(\)](#)
- [sss_key_store_generate_key\(\)](#)
- [sss_se05x_key_store_get_key_attst\(\)](#)
- [kAlgorithm_SSS_ECDSA_SHA256 from sss_algorithm_t](#)
- [kMode_SSS_Verify from sss_mode_t](#)
- [sss_asymmetric_verify_digest\(\)](#)

5.2.8.3 Console output

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

```
App :INFO :Running ECC key attestation example ex_sss_attest_ecc.c
App :INFO :Inject ECC key pair - 'ecc_key'
App :INFO :Create a attestation ECC key pair - 'attestation_ecc_key'
App :INFO :Read public key from ECC key pair 'ecc_key' with attestation
App :INFO :Attested Key -->
App :INFO :Public Key (Len=91)
      30 59 30 13    06 07 2A 86    48 CE 3D 02    01 06 08 2A
      86 48 CE 3D    03 01 07 03    42 00 04 ED    A7 E9 0B F9
```

```

20 CF FB 9D      F6 DB CE F7      20 E1 23 8B      3C EE 84 86
D2 50 E4 DF      30 11 50 1A      15 08 A6 2E      D7 49 52 78
63 6E 61 E8      5F ED B0 6D      87 92 0A 04      19 14 FE 76
63 55 DF BD      68 61 59 31      8E 68 7C
App :INFO :Attribute (Len=17)
EF 00 00 92      01 01 00 00      00 00 00 00      00 00 01 00
00
App :INFO :Time Stamp (Len=12)
00 00 00 0F      FF FF FF FA      00 01 2C C8
App :INFO :Out Random Value (Len=16)
01 02 03 04      05 06 07 08      09 01 02 03      04 05 06 07
App :INFO :Chip Id (Len=18)
04 0D 0F 00      39 67 C2 C1      CE 41 1B 9A      76 F5 B7 AB
D0 CD
App :INFO :Signature (Len=71)
30 45 02 21      00 8E 09 C2      BD 27 45 7B      0A 6E 2C 70
48 00 B0 D7      AC 7D FA 70      03 7B 21 A5      BF 0B 40 8B
C8 99 76 B0      03 02 20 6F      F0 22 37 24      43 22 0C 7B
99 2A DA DA      0D D6 BF AD      A6 B3 09 81      0A 2D 0A BD
5D 59 17 39      EC 24 93
App :INFO :Verify attestation signature using 'attestation_ecc_key' key
App :INFO :Verification success
App :INFO :ex_sss_attest_ecc Example Success !!!...
App :INFO :ex_sss Finished

```

5.2.9 ECC MONTGOMERY-25519 Key Attestation Example

This project demonstrates ecc montgomery25519 key attestation and verification with ecc nist256 key using SSS API. Signing on montgomery public key is done with key in the big endian format inside secure element.

Refer - simw-top/sss/ex/attest_mont/ex_sss_mont_attest.c

Note:

For twisted edward curve and montgomery 448 curve also the attestation signing is done with public key in big endian format.

5.2.9.1 Prerequisites

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

5.2.9.2 About the Example

This example reads a montgomery-25519 public key with attestation.

It uses the following APIs and data types:

- [`sss_key_store_set_key\(\)`](#)
- [`sss_key_store_generate_key\(\)`](#)
- [`sss_se05x_key_store_get_key_attst\(\)`](#)
- [`kAlgorithm_SSS_ECDSA_SHA256`](#) from [`sss_algorithm_t`](#)
- [`kMode_SSS_Verify`](#) from [`sss_mode_t`](#)
- [`sss_asymmetric_verify_digest\(\)`](#)

5.2.9.3 Console output

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

```

App :INFO :Running ECC key attestation example ex_sss_attest_mont.c
App :INFO :Inject ECC Montgomery key pair - 'ecc_mont_key'
sss :WARN :nxEnsure:'ret == SM_OK' failed. At Line:5621 Function:sss_se05x_TXn
App :INFO :Create a attestation ECC key pair - 'attestation_ecc_key'
App :INFO :Read public key from ECC key pair 'ecc_key' with attestation
App :INFO :Attested Key -->
App :INFO :Public Key (Little endian format) (Len=44)
 30 2A 30 05 06 03 2B 65 6E 03 21 00 85 20 F0 09
 89 30 A7 54 74 8B 7D DC B4 3E F7 5A 0D BF 3A 0D
 26 38 1A F4 EB A4 A9 8E AA 9B 4E 6A
App :INFO :Attribute (Len=17)
EF 00 00 8A 01 01 00 00 00 00 00 00 01 00
00
App :INFO :Time Stamp (Len=12)
00 00 00 18 FF FF FF ED 00 01 30 B0
App :INFO :Out Random Value (Len=16)
01 02 03 04 05 06 07 08 09 01 02 03 04 05 06 07
App :INFO :Chip Id (Len=18)
04 0D 0F 00 39 67 C2 C1 CE 41 1B 9A 76 F5 B7 AB
D0 CD
App :INFO :Signature (Len=71)
30 45 02 20 7A F8 E5 21 E7 0D 01 A9 9A AC 99 2A
AA 36 3A 89 C0 A7 CE 43 2B 85 CC FB 6E 98 A1 9C
2A E8 BC 4B 02 21 00 E7 0F 82 DD F2 43 C3 B2 CD
54 26 61 F0 06 AE 98 5B 2C 95 89 D7 A2 F3 4A 6E
13 12 A4 82 23 FF BD
App :INFO :Verify attestation signature using 'attestation_ecc_key' key
App :INFO :Singing is done on public key without header and with key in big
  endian format
App :INFO :Covert the key to big endian format for verification of signature
App :INFO :Verification success
App :INFO :ex_sss_attest_mont Example Success !!!...
App :INFO :ex_sss_Finished

```

5.2.10 EDDSA Example

This project demonstrates EDDSA sign and verify operation using SSS APIs. Sign and Verify operations using edwards key should to be performed on plain data. Use `sss_se05x_asymmetric_sign` and `sss_se05x_asymmetric_verify` apis.

Refer - `simw-top/sss/ex/eddsa/ex_sss_eddsa.c`

5.2.10.1 Prerequisites

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

5.2.10.2 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_se05x_asymmetric_sign\(\)](#)
- kMode_SSS_Verify from [sss_mode_t](#)
- [sss_se05x_asymmetric_verify\(\)](#)

5.2.10.3 Console output

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

```
App :INFO :Running EDDSA Example ex_sss_eddsa.c
App :INFO :Do Signing
App :INFO :Source Data (Len=50)
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
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00
App :INFO :signature (Len=64)
7E 88 E2 90 3A 2E 58 2B 40 30 1E D9 E6 38 D7 D1
C4 E3 43 AF 1D F5 6B A4 FF CE 38 9A 92 EB 39 F6
E9 8E F8 AD 54 51 B8 76 58 5D 2D 94 2F 46 EB B6
58 E3 32 BC 84 C2 A3 14 49 C8 F5 1A C9 98 3D 0E
App :INFO :Signing Successful !!!
App :INFO :Do Verify
App :INFO :Source Data (Len=50)
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
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00
App :INFO :signature (Len=64)
7E 88 E2 90 3A 2E 58 2B 40 30 1E D9 E6 38 D7 D1
C4 E3 43 AF 1D F5 6B A4 FF CE 38 9A 92 EB 39 F6
E9 8E F8 AD 54 51 B8 76 58 5D 2D 94 2F 46 EB B6
58 E3 32 BC 84 C2 A3 14 49 C8 F5 1A C9 98 3D 0E
App :INFO :Verification Successful !!!
App :INFO :ex_sss_eddsa Example Success !!!...
App :INFO :ex_sss Finished
```

5.3 AWS Demo for MCU-SDK

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

The example demonstrates standard flow of AWS cloud connectivity (Not JITR flow).

5.3.1 Prerequisites

- Active AWS account
- MCUXpresso installed (for running aws demo on K64F)
- Any Serial communicator
- Flash VCOM binary on the device. VCOM binary can found under `simw-top/binaries/MCU` folder.
- Refer to [CLI Tool](#) for `ssscli` tool setup
- Build Plug & Trust middleware stack. (Refer [Building / Compiling](#))

5.3.2 Using WiFi with LPC55S

WiFi shield CMWC1ZZABR-107-EVB by muRata is supported with LPC55S. Mount the WiFi shield on to the mikroBUS stackable headers.

Connection of muRata with LPC55s

Refer - <https://www.murata.com/en-sg/products/connectivitymodule/wi-fi-bluetooth/overview/lineup/typeabr>

Update `clientcredentialWIFI_SSID` and `clientcredentialWIFI_PASSWORD` in `simw-top/demos/ksdk/common/wifi_config.h` with your wifi credentials.

5.3.3 Creating a device on AWS account

Refer - <https://docs.aws.amazon.com/iot/latest/developerguide/iot-gs-first-thing.html>

Refer - <https://aws.amazon.com/blogs/mobile/use-your-own-certificate-with-aws-iot/#:~:text=In%20the%20AWS%20Console%2C%20navigate%20to%20AWS%20IoT%2C,to%20upload%20your%20device%20certificate%20into%20AWS%20IoT>

Refer - <https://www.nxp.com/docs/en/application-note/AN12404.pdf>

5.3.4 Creating and updating device keys and certificates to SE

1. To generate AWS credentials, refer [AWS IoT Credentials](#) Ensure that the appropriate Root CA, Root Ca Verification Certificate and device certificate are uploaded to AWS.
2. Complete [Section 9.3 Steps needed before running ssscli tool](#)
3. Flash the vcom binary from `simw-top/binaries/MCU` and check the vcom port number
4. Provision the generated device key in SE050 using ssscli.

```
ssscli connect SE05X <connection method> <port name>
ssscli set ecc pair <key-id> <device-key>
ssscli set cert <cert-id> <device-certificate>
```

1. Update the `SSS_CERTIFICATE_INDEX_CLIENT` and `SSS_KEYPAIR_INDEX_CLIENT_PRIVATE` in `aws_iot_config.h`

5.3.5 Building the Demo

1. Open cmake project found under `SIMW-TOP/projects` in MCUXPRESSO IDE
2. To get the AWS IoT MQTT broker endpoint for your account, go to the AWS IoT console and in the left navigation pane choose Settings. Copy the endpoint listed under the “Device data endpoint”
3. In the `simw-top/demos/ksdk/common/aws_client_credential_keys.h` file, update the endpoint in the macro “`clientcredentialMQTT_BROKER_ENDPOINT`” and thing name in the macro “`clientcredentialIOT_THING_NAME`”
4. In the `simw-top/demos/ksdk/common/aws_client_credential_keys.h` file, update the endpoint in the macro “`clientcredentialMQTT_BROKER_ENDPOINT`” and thing name in the macro “`clientcredentialIOT_THING_NAME`”
5. **Update cmake options::**
 - RTOS=FreeRTOS

- mbedTLS_ALT=SSS

6. Update cmake options for Host MCXN947::

- RTOS=FreeRTOS
- mbedTLS=3_X

7. Update the build target in make file

- Project:cloud_aws

5.3.6 Running the Demo

1. 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

2. 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.

3. Download the program to the target board.

4. Either press the reset button on your board or launch the debugger in your IDE to begin running the demo.

5. The BLUE LED is turned ON during boot

6. Persistent RED LED ON indicates error

7. First time during connection, the device certificate needs to be

- Activated
- Attached with a policy that allows usage of this certificate

8. 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

- To generate AWS credentials, refer [AWS IoT Credentials](#) Ensure that the appropriate Root CA, Root Ca Verification Certificate and device certificate are uploaded to AWS.
- Complete [Section 9.3 Steps needed before running ssscli tool](#)
- Provision the generated device key in SE050 using ssscli.

```
ssscli connect se05x <connection method> <port name>
ssscli set ecc pair <key-id> <device-key>
```

- Generate the reference key

```
ssscli refpem ecc pair <key-id> <output-ref-key>
```

5.4.2.1 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_t1oi2c
cmake -DPTMW_OpenSSL:STRING=1_0 -DPTMW_HostCrypto:STRING=OPENSSL .
cmake --build .
make install
ldconfig /usr/local/lib
```

Note:

Replace imx_native_se050_t1oi2c with raspbian_native_se050_t1oi2c when building for Raspberry Pi.

5.4.2.2 Build the OpenSSL provider [Optional]

Note:

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

The OpenSSL provider 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 provider for SE050 connected over I2C.

```
cd simw-top
python scripts/create_cmake_projects.py
cd ../simw-top_build/imx_native_se050_t1oi2c
cmake -DPTMW_OpenSSL:STRING=3_0 -DPTMW_HostCrypto:STRING=OPENSSL .
cmake --build .
make install
```

```
ldconfig /usr/local/lib
```

Note:

Replace `imx_native_se050_t1oi2c` with `raspbian_native_se050_t1oi2c` 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
```

2. For the demo to build with openssl 3.0 and to use sssProvider apply the patch to repo aws-iot-device-sdk-cpp:

```
cd /simw-top/demos/linux/aws/aws-iot-device-sdk-cpp  
patch -p1 < ../sssProvider_Updates.patch
```

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

3. 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 ()
```

4. Convert the `buildScript.sh` file from DOS (or Windows) format to Unix format and set the permissions:

```
chmod 777 buildScript.sh  
dos2unix buildScript.sh
```

5. Now run ‘`buildScript.sh`’ script at `simw-top/demos/linux/aws/` to build the mqtt application for aws call:

```
./buildScript.sh
```

6. 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 “xxxxiukfoyyyy-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_ticks_to_wait_for": 32,
    "action_processing_rate_hz": 5,
    "maximum_outgoing_action_queue_length": 32,
    "discover_action_timeout_msecs": 300000
}

```

7. For OpenSSL 1.x, 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

```

8. For OpenSSL 3.x, search for *provider_sect* in /simw-top/demos/linux/common/openssl130_sss_se050.cnf file and set the nxp provider with high priority.

```

nxp_prov = nxp_prov_sec
default = default_sect

```

9. 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_se050.cnf      ----- OpenSSL 1.1.1 and SE050
openssl_sss_se050.cnf        ----- OpenSSL 1.0.0 and SE050
openssl130_sss_se050.cnf     ----- OpenSSL 3.x and SE050

```

10. Set the openssl config path as call:

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

11. 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 Azure Demo for MCU-SDK

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

The example is last tested with Plug and Trust Middleware version 4.5.1.

5.5.1 Prerequisites

- Active azure account with iot hub created
- MCUXpresso installed (for running azure demo on K64F)
- Any Serial communicator
- Flash VCOM binary on the device. VCOM binary can found under <PROJECT>/binariesMCU folder.
- Refer to [CLI Tool](#) for ssscli tool setup
- Build Plug & Trust middleware stack. (Refer [Building / Compiling](#))

5.5.2 Using WiFi with LPC55S

WiFi shield CMWC1ZZABR-107-EVB by muRata is supported with LPC55S. Mount the WiFi shield on to the mikroBUS stackable headers.

Connection of muRata with LPC55S

Refer - <https://www.murata.com/en-sg/products/connectivitymodule/wi-fi-bluetooth/overview/lineup/typeabr>

1. Update clientcredentialWIFI_SSID and clientcredentialWIFI_PASSWORD in simw-top/demos/ksdk/common/wifi_config.h with your wifi credentials.

5.5.3 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

5.5.4 Creating and updating device keys and certificates to SE

1. Complete [Section 9.3 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>
```

Note:

Provisioning of the keys is done with default policies. Refer - [Section 9.9](#) to change the scripts to add required policies.

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

5.5.5 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

5.5.6 Building the Demo

1. Open cmake project found under SIMW-TOP/projects in MCUXPRESSO IDE
2. Update AZURE_IOT_HUB_NAME and AZURE_DEVICE_NAME in simw-top/demos/ksdk/azure/azure_iot_config.h with your details.
3. Update cmake options::
 - RTOS=FreeRTOS
 - mbedTLS_ALT=SSS
4. Update cmake options for Host MCXN947::
 - RTOS=FreeRTOS
 - mbedTLS=3_X
5. Update the build target in make file
 - Project:cloud_azure

5.5.7 Running the Demo

1. Update AZURE_IOT_HUB_NAME and AZURE_DEVICE_NAME in <PROJECT>\demo\azure_demo\azure_iot_config.h with your details
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

- 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 1X
| 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"}]
```

5.5.8 Appendix

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

5.6 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.

The example is last tested with Plug and Trust Middleware version 4.5.1.

5.6.1 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
 - On Raspberry Pi, use the commands to ensure that the setup is complete.
 1. sudo apt-get update
 2. sudo apt-get install autoreconf
 3. sudo apt-get install libtool
 4. sudo apt-get install build-essential gcc make cmake cmake-gui cmake-curses-gui

5.6.2 Preparing the credentials and Provisioning the secure element

Use ssscli tool from iMX/Rpi platform

1. Complete [Section 9.3 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
```

Note:

Provisioning of the keys is done with default policies. Refer - [Section 9.9](#) to change the scripts to add required policies.

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

5.6.3 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](#).

5.6.4 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 alphanumerics 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 \[Optional\]](#).

5.6.5 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

5.6.6 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_t1oi2c
cmake --build .
make install
ldconfig /usr/local/lib
```

Note:

Replace `imx_native_se050_t1oi2c` with `raspbian_native_se050_t1oi2c` when building for Raspberry Pi.

5.6.6.1 Build the OpenSSL provider [Optional]

Note:

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

The OpenSSL provider 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 provider for SE050 connected over I2C.

```
cd simw-top
python scripts/create_cmake_projects.py
cd ..simw-top_build/imx_native_se050_t1oi2c
cmake -DPTMW_OpenSSL:STRING=3_0 -DPTMW_HostCrypto:STRING=OPENSSL .
cmake --build .
make install
ldconfig /usr/local/lib
```

Note:

Replace `imx_native_se050_t1oi2c` with `raspbian_native_se050_t1oi2c` when building for Raspberry Pi.

5.6.7 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. Search for `provider_sect` in /simw-top/demos/linux/common/openssl130_sss_se050.cnf file and set the nxp provider with high priority.

```
nxp_prov = nxp_prov_sec default = default_sect
```

3. 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_se050.cnf      ----- OpenSSL 1.1.1 and SE050
openssl_sss_se050.cnf        ----- OpenSSL 1.0.0 and SE050
openssl130_sss_se050.cnf    ----- OpenSSL 3.x and SE050
```

4. Set the openssl config path as call:

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

5. To run the application, call:

```
$ ./azure_imx_connect --deviceid "<devide_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 username

5.7 OpenSSL Engine / Provider: 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 / OpenSSL Provider on the client side.

5.7.1 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 key material (CA, Client and Server) to be either RSA (4096 CA - 2048 client/server) or EC (prime256v1).

The TLS demo comes in two flavors:

1. **Flavor-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. **Flavor-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 5.4.1.2](#) to [Section 5.4.1.5](#) are identical for the two demo flavours.

5.7.2 Credential preparation (execute once) [Optional]

```
> cd demos/linux/tls_client
> chmod +x scripts/createTlsCredentials_Optional.sh
> ./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.

5.7.3 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:

5.7.3.1 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 => t1oi2c, sci2c, vcom, jrcpv1, jrcpv2, pcsc. Default: t1oi2c
--subsystem SUBSYSTEM	Supported subsystem => se050. 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
```

5.7.4 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 5.4.1.2](#)) are available on the server platform.

5.7.5 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  
> chmod +x tlsServer.sh  
> ./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  
> chmod +x tlsServer.sh  
> ./tlsServer.sh RSA
```

5.7.6 Establish a TLS link from the client to the server

The client process establishing the TLS connection comes in two flavors: either *s_client* or a program provided in source code (*tlsSe050Client.cpp*). Invoke either example through a bash shell script.

5.7.6.1 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:

```
> chmod +x tlsSeClient.sh  
> ./tlsSeClient.sh <server-IP-address> <ECDHE|ECDHE_256>  
OR  
> chmod +x tlsSeClient_3_0.sh  
> ./tlsSeClient_3_0.sh <server-IP-address> <ECDHE|ECDHE_256> # For TLS using  
OpenSSL 3.0 Provider
```

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:

```
> chmod +x tlsSeClient.sh  
> ./tlsSeClient.sh <server-IP-address> RSA  
OR
```

```
> chmod +x tlsSeClient_3_0.sh  
> ./tlsSeClient_3_0.sh <server-IP-address> RSA      # For TLS using OpenSSL 3.0  
Provider
```

In case OpenSSL 1.1.1 / OpenSSL 3.x 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
```

5.7.6.2 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:

```
> chmod +x tlsExtendedSeClient.sh  
> ./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 / OpenSSL 3.x 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.

5.8 Access Manager: Manage access from multiple (Linux) processes to an SE05x IoT Applet

- DocRevision : 0.93
- Date : 2020-10-20

5.8.1 Summary

The Access Manager supports concurrent access from multiple linux processes to an SE05x IoT applet. The Access Manager can establish a connection to the SE05x either as a plain connection or as an SCP03 platform connection. Client processes connect over the JRCPv1 protocol to the accessManager. Refer to [Concepts & Features](#) for more details.

Note:

When using multiple clients with user authentication, build client applications with -DPTMW_SMCOM:STRING=JRCP_V1_AM option. This will ensure the user session is established in an atomic way per client connect.

Note:

On Error 0x66A6 (ThroughPut error), access manager will do a soft reset of SE and will re-establish the platformSCP session.

5.8.2 Usage

```
The accessManager takes two optional arguments 'plain' & 'any'
'accessManager':
    Platform SCP03: ON.
    Incoming connection: localhost.
'accessManager plain':
    Platform SCP03: OFF.
    Incoming connection: localhost.
'accessManager any':
    Platform SCP03: ON.
    Incoming connection: any supported address.
'accessManager plain any':
    Platform SCP03: OFF.
    Incoming connection: any supported address.
```

Note:

Product Deployment => Enable Platform SCP03 & restrict incoming connection to localhost

In case STREAM sockets are used client processes must connect to port 8040.

As an example:

- The Access Manager is running on iMX and opens a listening STREAMING socket. Incoming connections can be restricted to client processes connecting over localhost.

Example invocation. Notice that the Platform SCP03 keys are passed through an environment variable:

```
root@imx8mqevk:~/mnt/git/simw-top_build_imx8_5_4_24/imx_native_se050_t1o12c# \
EX_SSS_BOOT_SCP03_PATH=/home/root/plain_scp03.txt bin/accessManager
Starting accessManager (Rev.0.9).
Protect Link between accessManager and SE: YES.
accessManager JRPCv1 (T1oI2C SE side)
*****
Server: waiting for connections on port 8040.
Server: only localhost based processes can connect.
```

- client process connects via JRPCv1 to 127.0.0.1:8040

Example invocation. Notice that the server address is set through an environment variable. In a product deployment the default server:port address can also be hard-coded to the proper value:

```
root@imx8mqevk:~/home/root# EX_SSS_BOOT_SSS_PORT=127.0.0.1:8040
se05x_ConcurrentEcc
```

5.8.3 Build

- The Access Manager must be built as a statically linked executable as its communication and authentication layer is different from the client processes that connect to it.
- Access manager will send “READY=1” start up notification to service manager. This is disabled by default. To enable, uncomment below lines in simw-top/hostlib/hostLib/accessManager/CMakeLists.txt file

```
ADD_DEFINITIONS(-DENABLE_SD_NOTIFY)
TARGET_LINK_LIBRARIES(${PROJECT_NAME} systemd)
```

Note:

To build access manager on RaspberryPi with ENABLE_SD_NOTIFY enabled, install systemd-dev lib as
`sudo apt-get install libsystemd-dev`

Build settings to support access to SE05x on iMX host platform (to be applied on top of a configured host build area):

```
cmake -DPTMW SCP:STRING=SCP03_SSS -DPTMW_SE05X_Auth:STRING=PlatfSCP03 -  
DPTMW_SMCOM:STRING=T1oI2C \  
-DWWithSharedLIB:BOOL=OFF -DPAHO_BUILD_SHARED:BOOL=FALSE -  
DPAHO_BUILD_STATIC:BOOL=TRUE .  
cmake --build . --target accessManager
```

- The client processes that connect to the Access Manager must be built in a separate build environment. All session authentication mechanisms are supported, platform SCP03 must be off (platform SCP03 is handled by the Access Manager).

Build settings for client processes connecting via Access Manager, in the example no session authentication is used (to be applied on top of a configured host build area):

```
cmake -DPTMW_SE05X_Auth:STRING=None -DPTMW_SMCOM:STRING=JRPC_V1_AM .  
cmake --build .
```

5.8.4 Demo: concurrent access from 2 processes using OpenSSL engine

- The example requires an embedded Linux platform (e.g. an iMX8) with an attached SE05X. Interaction with the iMX8 is over 3 different shells. These shells can e.g. be established via ssh from a PC on the same network.
- Build the Access Manager in a dedicated work area, follow build instructions as above. Select static linking, enable Platform SCP03 and use T1oI2C as communication protocol.
- Build the Plug&Trust package in a dedicated work area, follow build instructions as above. Select None as authentication mode and use JRPCv1 as communication protocol.
- Start the Access Manager from a dedicated shell (to simplify the demo, Platform SCP03 is not enabled):

```
./accessManager plain
```

- Open another shell and configure the attached Secure Element once using the ssscli tool (ensure the installed ssscli tool uses JRPCv1 as communication protocol, refer to [Communication interface \(cmake SMCOM setting\)](#)):

```
cd <plug_and_trust>/simw-top/sss/plugin/openssl/scripts  
python3 openssl_provisionEC.py --key_type prime256v1 --connection_data  
127.0.0.1:8040
```

- From the same shell invoke the OpenSSL Engine to perform various sign/verify operations using the provisioned EC key pairs:

```
python3 openssl_EccSign.py --key_type prime256v1 --connection_data  
127.0.0.1:8040
```

- Open another shell and invoke the OpenSSL Engine to perform various sign/verify operations using the provisioned EC key pairs:

```
cd <plug_and_trust>/simw-top/sss/plugin/openssl/scripts  
python3 openssl_EccSign.py --key_type prime256v1 --connection_data  
127.0.0.1:8040 --output_dirname output3
```

- The respective 'openssl_EccSign.py' invocations can be repeated, ensure both process invocations run in parallel.

5.8.5 Example programs prepared for concurrent access

The demo folder of the Plug&Trust MW package contains two SSS API based example programs that are compatible with concurrent access requirements like:

- ability to select a specific (optional) authentication object ID
- provisioned content of secure element is not erased at project start-up

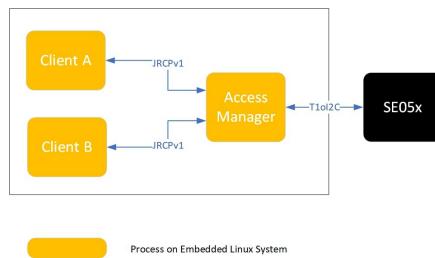
For more details on these examples refer to:

- [Section 5.7.25ECC Concurrent Example](#)
- [Section 5.7.26Symmetric Multi Step Concurrent Example](#)

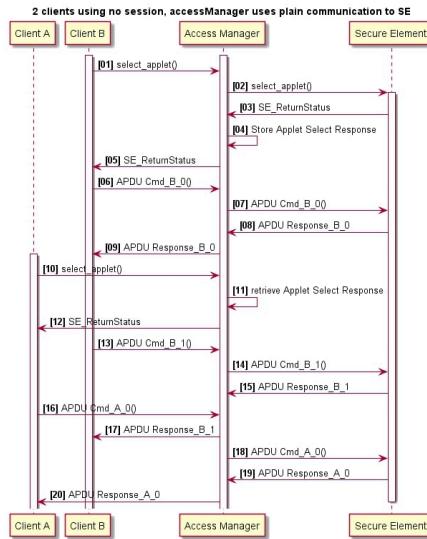
5.8.6 Concepts & Features

- The Access Manager uses plain communication or platform SCP03 in the communication with the SE. Select the mode at start-up.
- Client processes connect to the accessManager using the JRCPv1 protocol
- The user session authentication type is determined at the client build time. User session authentication is transparent to the Access Manager.
- The Access Manager ensures APDU command / response pairs associated with a client process are executed without interference from another client process.
- The Access Manager does not connect to the SE05x at start up. It waits until a client process initiates a connection.
- When a client process selects the SE05x IoT applet the applet response is cached by the Access Manager, a subsequent SE05x IoT applet select by a client process will simply return the cached applet response.
- A card manager select command is intercepted by the Access Manager and a pre-cooked response is provided to the initiating client process. No interaction with the secure element takes place.

The following figure illustrates the Access Manager is an independent process on the Embedded System providing indirect access to the Secure Element for client processes.



The following sequence diagram illustrates two processes connecting through the Access Manager to the Secure Element.



5.8.7 Restrictions

- Each user session needs to have a different authentication object; i.e. one Authentication Object cannot be used to open multiple sessions in parallel. This limitation is inherent to the SE05x user session concept.
- The SE05x does not support more than two active user sessions (based upon either a User ID, AES Key or EC Key authentication object). The Access Manager does not and - conceptually - cannot monitor the number of active user sessions.
- The Access Manager only supports concurrent access to the SE05x IoT applet. Do not access other applets than the SE05x applet through the Access Manager.
- The Access Manager does not attempt to re-establish a broken connection to the SE05x. To recognize and recover from a broken connection, a system integrator must monitor failure to communicate to the Secure Element by the client processes. As and if required the Access Manager must be restarted and the affected client processes must reconnect to the Access Manager.
- A client process establishing a user session with the SE05x applet must always close the user session prior to disconnecting from the Access Manager.
- Selecting another applet than the SE05x IoT applet is possible but strongly discouraged and not supported.
- The Access Manager **does not** :
 - Handle power management
 - Keep track of Secure Element resources
- In a typical deployment the Access Manager and client processes are controlled by another – product specific - entity on the Embedded System:
 - In case of an applet update, the Access Manager must be shut down and control of the secure element must be handed over to the SEMS Lite update manager.
 - A credential update must be coordinated between the consuming processes and the updating process. Such coordination is out-of-scope of the Access Manager
- Transparent usage of the OpenSSL Engine from different applications implies either no user session (Auth=None) or using the OpenSSL Engine from isolated environments (with different authentication settings). This restriction does not apply to applications built directly on top of the SSS API.
- The SSS layer's implementation of multi-step symmetric ciphers does not allow concurrent execution of ciphers with the same cipher mode (e.g. twice kAlgorithm_SSS_AES_CBC).

5.8.8 Session open retry in client processes (Only for SE05X)

Session open retries can be enabled in client processes if required. (Only with – DPTMW_SMCOM:STRING=JRCP_V1_AM build option)

Set env variable EX_SSS_SESSION_OPEN_RETRY_CNT to number of retries required. (Default - 1. Max - 50).

Set env variable EX_SSS_SESSION_OPEN_RETRY_DLY to delay between each retries. (Default - 1 second. Max - 10 seconds).

5.8.9 Access Manager with Unix sockets

By default stream sockets are enabled for access manager. To enable Unix sockets in access manager, build using below option (to be applied on top of a configured host build area):

```
cmake -DWithAccessMgr_UxSocket:BOOL=ON .
cmake --build . --target accessManager
```

Access manager on start up will create unix socket file at - /var/run/am.

When using access manager with Unix sockets, build client applications also with WithAccessMgr_UxSocket:BOOL=ON option

Note:

Product Deployment => Assign the access permissions of /var/run/am accordingly.

Note:

Product Deployment => If you wish to change the unix socket file location of access manager, modify the define 'UNIX_SOCKET_FILE' in simw-top/hostlib/hostLib/accessManager/inc/accessManager.h.

Also change the default socket port for client applications by changing - EX_SSS_BOOT_SSS_SOCKETPORT_DEFAULT in simw-top/sss/ex/inc/ex_sss_ports.h

5.8.10 Access Manager command to reset secure element

SM_AmResetI2C API in the SMCOM layer can be used to send the reset request to access manager. On receiving this command, access manager will do the T1oI2C reset using phNxpEse_reset API and re-establishes PlatfSCP03 session if required.

Usage - Include sm_api.h header file in the application and call SM_AmResetI2C API. The API needs the connection context as input argument. This can be passed from session context. Refer the below code snippet

```
sss_se05x_session_t *pSession = (sss_se05x_session_t *)<SSS_SESSION>;
Se05xSession_t *pSe05xSession = &pSession->s_ctx;
...
SM_AmResetI2C(pSe05xSession->conn_ctx);
```

Return value - success ==> 0, Failure ==> 1

Reset Command / response format

- Reset Command Format : [0x60, 0x00, 0x00, 0x00], 0x60 is Access Manager reset command MTY_T1OI2C_RESET
- Reset Response Format: [0x60, 0x00, 0x00, 0x00/0x01], In case of success 0x00 While 0x01 returned as a last byte in case of any failure.

5.9 OPC UA (Open62541) Demo

Note:

The example here is only for demonstration of SE05x integration. Do not use the same in product deployment. The example is last tested with Plug and Trust Middleware version 4.5.1.

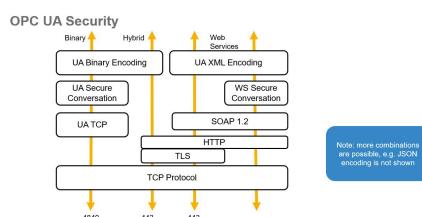
5.9.1 Supported Platforms

- Server Platform
 - Windows – JRCPv2 – SE050
 - iMX6 / RaspberryPi - t1oi2c – SE050
- Client Platform
 - UaExpert on Windows
 - Open62541 client on Windows
 - Open62541 client on iMX6 / RaspberryPi

5.9.2 Introduction

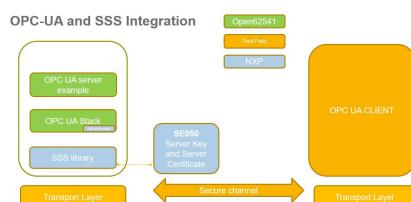
OPC UA (Open Platform Communications Unified Architecture) is an application layer protocol specific to Industrial IoT. It can run on top of TCP, TCP + Web services or TCP + HTTPS. In this client - server demo, the Open62541 open source OPC UA stack is used for integration with SE050. The server certificate 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:



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



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

5.9.3 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 -
DWithSharedLIB:BOOL=OFF .
cmake --build .
make install
ldconfig /usr/local/lib
```

Note:

Replace `imx_native_se050_t1oi2c` with `raspbian_native_se050_t1oi2c` when building for Raspberry Pi.

1. Server and client binaries are copied to the simw-top/tools folder

5.9.4 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 `ssscli` tool setup. Using `ssscli` 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.**
 1. UaExpert -> Settings -> Configure UaExpert -> General.DisableError.CertificateIssuerRevocationUnknown -> true
 2. UaExpert -> Settings -> Configure UaExpert -> General.DisableError.CertificateRevocationUnknown -> true
- **Add the server details to connect. UaExpert -> Server -> Add -> Advanced (Tab). Add details in**
 1. EndPoint Url (opc.tcp://<SERVER_IP>:4840/)
 2. Security Policy as Basic256Sha256
 3. 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.

5.9.5 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.

5.10 PSA Non Secure Example

This example is to demonstrate how to use PSA library APIs to perform a Sign/Verify operations.

5.10.1 Pre-requisites

You need to build PSA-ALT library for TrustZone before compiling this application code.

Refer to [Section 8.5Platform Security Architecture](#).

5.10.2 PSA Operation Examples

- Generating asymmetric keys.

```

psa_key_id_t key_id           = PSA_ALT_ITS_SE_FLAG | 0x00181001;
psa_key_attributes_t attributes = PSA_KEY_ATTRIBUTES_INIT;
psa_set_key_usage_flags(&attributes,
    PSA_KEY_USAGE_ENCRYPT | PSA_KEY_USAGE_DECRYPT | PSA_KEY_USAGE_EXPORT |
    PSA_KEY_USAGE_SIGN_HASH |
    PSA_KEY_USAGE_VERIFY_HASH);
psa_set_key_algorithm(&attributes, alg);
psa_set_key_type(&attributes, PSA_KEY_TYPE_RSA_KEY_PAIR);
psa_set_key_bits(&attributes, key_bits);
psa_set_key_lifetime(&attributes, lifetime);
psa_set_key_id(&attributes, key_id);

LOG_I("Generating RSA-%d key", key_bits);

status = psa_generate_key(&attributes, &key_handle);

```

- Performing Sign-Verify operations.

```

uint8_t hash[32]      = {1};
size_t hashLen       = sizeof(hash);
uint8_t signature[256] = {0};
size_t sigLen         = sizeof(signature);

status = psa_sign_hash(
    key_handle, PSA_ALG_RSA_PKCS1V15_SIGN(PSA_ALG_SHA_256), hash, hashLen,
    signature, sizeof(signature), &sigLen);
if (status != 0) {
    LOG_E("Signing failed");
    goto cleanup;
}
else {
    LOG_I("Signing success");
}

status = psa_verify_hash(key_handle, PSA_ALG_RSA_PKCS1V15_SIGN(PSA_ALG_SHA_256),
    hash, hashLen, signature, sigLen);
if (status != 0) {
    LOG_E("Verification failed");
    goto cleanup;
}
else {
    LOG_I("Verification success");
}

```

5.10.3 Building Example

This example would run in normal world and must link with the secure world PSA library so that definitions for veneer APIs can be found. Build this example with the following CMake configurations:

- Host=lpcxpresso55s_ns
- HostCrypto=MBEDTLS
- mbedtls_ALT=PSA
- RTOS=Default
- SMCOM=T1oI2C

- PROJECT=psa_nonsecure

5.11 SE05X Minimal example

This project gets available memory from secure element. This is a good starting point to work with SE05X at low level.

5.11.1 Building the Demo

- Build Plug & Trust middleware stack. (Refer [Building / Compiling](#))
- Project: se05x_Minimal

5.11.2 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.

On Raspberry-Pi or iMX board, run as:

```
./se05x_Minimal
```

5.11.3 Console output

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

```
App :INFO :mem=32767
App :INFO :ex_sss Finished
```

5.12 SE05X Get Info example

This project can be used to get SE05X platform information with or without applet installed.

5.12.1 Building the Demo

- Build Plug & Trust middleware stack. (Refer [Building / Compiling](#))
- Project: se05x_GetInfo

5.12.2 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_GetInfo.exe <PORT NAME>
```

Where <PORT NAME> is the VCOM COM port.

On Raspberry-Pi or iMX board, run as:

```
./se05x_GetInfo
```

5.12.3 Console output

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

```
App :INFO :PlugAndTrust_v04.02.00_20220524
App :INFO :Running C:\_ddm\simw-top_build\se_x86\bin\Debug\se05x_GetInfo.exe
App :INFO :If you want to over-ride the selection, use
ENV=EX_SSS_BOOT_SSS_PORT or pass in command line arguments.
App :WARN :#####
App :INFO :uid (Len=18)
A0 A1 A2 A3      A4 A5 A6 A7      A8 A9 AA AB      AC AD AE AF
B0 B1
App :INFO :Running C:\_ddm\simw-top_build\se_x86\bin\Debug\se05x_GetInfo.exe
App :INFO :If you want to over-ride the selection, use
ENV=EX_SSS_BOOT_SSS_PORT or pass in command line arguments.
sss :WARN :Communication channel is Plain.
sss :WARN :!!!Not recommended for production use.!!!
App :WARN :#####
App :INFO :Applet Major = 7
App :INFO :Applet Minor = 2
App :INFO :Applet patch = 0
App :INFO :AppletConfig = 3FFF
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 = FFFF
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 01 A8 78      80 10 29 0E      F3 33 19 C1
App :INFO :OEF ID (Len=2)
A8 78
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      39 42 34 31      31 31 30 30
30 30 30 30      30 30 30 30
App :INFO :JCOP Platform ID = J3R351029B411100
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 = 0x01
App :INFO :Tag ROM ID (Must be 0x08) = 0x08
App :INFO :ROM ID (Len=8)
      52 4F 4D 5F      5F 49 44
App :INFO :Status Word (SW) (Len=2)
      90 00
App :INFO :se05x_GetInfoPlainApplet Example Success !!!...
App :WARN :#####
App :INFO :cplc_data.IC_fabricator (Len=2)
      47 90
App :INFO :cplc_data.IC_type1 (Len=2)
      D3 21
App :INFO :cplc_data.Operating_system_identifier (Len=2)
      47 00
App :INFO :cplc_data.Operating_system_release_date (Len=2)
      00 00
App :INFO :cplc_data.Operating_system_release_level (Len=2)
      00 00
App :INFO :cplc_data.IC_fabrication_date (Len=2)
      74 74
App :INFO :cplc_data.IC_Serial_number (Len=4)
      6E 6E 6E 62
App :INFO :cplc_data.IC_Batch_identifier (Len=2)
      62 62
App :INFO :cplc_data.IC_module_fabricator (Len=2)
      00 00
App :INFO :cplc_data.IC_module_packaging_date (Len=2)
      00 00
App :INFO :cplc_data.ICC_manufacturer (Len=2)
      00 00
App :INFO :cplc_data.IC_embedding_date (Len=2)
      00 00
App :INFO :cplc_data.IC_OS_initializer (Len=2)
      57 58
App :INFO :cplc_data.IC_OS_initialization_date (Len=2)
      59 4E
App :INFO :cplc_data.IC_OS_initialization_equipment (Len=4)
      4E 4E 4E 4E
App :INFO :cplc_data.IC_personalizer (Len=2)
      00 00
App :INFO :cplc_data.IC_personalization_date (Len=2)
      00 00
App :INFO :cplc_data.IC_personalization_equipment_ID (Len=4)
      00 00 00 00
App :INFO :cplc_data.SW (Len=2)
      90 00
App :INFO :ex_sss Finished
```

5.13 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

5.13.1 How to use

Run the demo as apdu_player_demo [Input-file] <Port>

5.14 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 (SSS_HAVE_SE05X_AUTH_USERID) || (SSS_HAVE_SE05X_AUTH_USERID_PLATFSCP03) //  
UserID Session  
#define EX_LOCAL_OBJ_AUTH_ID EX_SSS_AUTH_SE05X_UserID_AUTH_ID  
#elif (SSS_HAVE_SE05X_AUTH_NONE) || (SSS_HAVE_SE05X_AUTH_PLATFSCP03) //No auth  
#define EX_LOCAL_OBJ_AUTH_ID EX_SSS_AUTH_SE05X_NONE_AUTH_ID  
#elif (SSS_HAVE_SE05X_AUTH_AESKEY) || (SSS_HAVE_SE05X_AUTH_AESKEY_PLATFSCP03) //  
AESKey  
#define EX_LOCAL_OBJ_AUTH_ID EX_SSS_AUTH_SE05X_APPLETSCP_AUTH_ID  
#elif (SSS_HAVE_SE05X_AUTH_ECKEY) || (SSS_HAVE_SE05X_AUTH_ECKEY_PLATFSCP03) //  
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.

5.14.1 Sign Policy

Create a policy set using the authentication object ID

```
/*Logic to pass sign & verify policy*/  
const int allow_sign = 1;  
const int allow_verify = 0;  
  
/* 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*/  
        .asymkey = {  
            /*Policy for sign*/  
            .can_Sign = allow_sign,  
            /*Policy for verify*/  
            .can_Verify = allow_verify,
```

```
/*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 = {
.nPolicies = 2,
.policies = { &key_withPol, &common }
};
/* doc:end:allow-policy-sign-part1 */

status = sss_key_object_init(&object, &pCtx->ks);
if (status != kStatus_SSS_Success) {
LOG_E("sss_key_object_init Failed!!!");
goto exit;
}

status = sss_key_object_allocate_handle(
&object, keyId, kSSS_KeyPart_Pair, kSSS_CipherType_EC_NIST_P, keylen,
kKeyObject_Mode_Persistent);
if (status != kStatus_SSS_Success) {
LOG_E("key_object_allocate_handle Failed!!!");
goto exit;
}

/* 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 */
```

5.14.2 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

```
#if SSS_HAVE_SE05X_VER_GTE_07_02
    uint8_t pcr_expected_value[] = { 0x87, 0xD3, 0xE3, 0x93, 0x19, 0x8F, 0x5C,
    0x80, 0xE0, 0xBC, 0x9B, 0xC9, 0x82, 0x00, 0x1F, 0xB0, 0xEE, 0x20, 0x1C, 0x27,
    0x0B, 0x6D, 0xC8, 0x84, 0x52, 0xE4, 0x13, 0xA3, 0x25, 0x56, 0x81, 0x75 };
#else
    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, 0xa, 0x86, 0xd, 0x0c, 0xa4, 0x14 };
#endif%%CRLF
SP%   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);%%CRLF
SP%   const sss_policy_u common = {
        .type = KPolicy_Common,
        .auth_obj_id = TST_LOCAL_OBJ_AUTH_ID,
        .policy = {
            .common = {
                .req_Sm = 0,
                .can_Delete = 1
            }
        }
    };%%CRLF
SP%   const sss_policy_u file = {
        .type = KPolicy_File,
        .auth_obj_id = TST_LOCAL_OBJ_AUTH_ID,
        .policy = {
            .file = {
                .can_Read = 1,
                .can_Write = 1
            }
        }
    };%%CRLF
SP%   sss_policy_u pcr1 = {
        .type = KPolicy_Common_PCR_Value,
        .auth_obj_id = TST_LOCAL_OBJ_AUTH_ID,
        .policy = {
            .common_pcr_value = {
                .pcrObjId = 0xffffffff,
            }
        }
    };
    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);%%CRLF
SP%   sss_policy_t policy_for_binary_object = {
```

```
.nPolicies = 3,  
.policies = { &common, &pcl1, &file }  
};  
/* clang-format on */  
NVM_RESET();%%CRLF  
SP%% 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.

5.14.3 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
```

5.15 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.

5.15.1 Building the example

Use the following CMake configurations to compile the example

- CMake configurations: SSS_HAVE_HOSTCRYPTOMBEDTLS: ON
- Project: se05x_GetCertificate

5.15.2 How to use

Run the tool as:

```
se05x_GetCertificate <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.

5.16 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.

Note:

SCP03 uses 3 sets of AES keys ENC MAC and DEK. After first rotation user can delete DEK key from file/code for security purpose.

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 se policy accordingly.

5.16.1 Prerequisites

Since this example is portable across various platforms, the needs are different.

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

5.16.2 Configuring the Demo

New Platform SCP03 keys are defined as following. Update your keys here.

```
#if ROTATE_PLATFORMSCP03_128BITLEN
#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 \
}
#else
#define EX_SSS_AUTH_NEW_ENC_KEY \
{
    \
    0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, 0x4A, 0x4B,
0x4C, 0x4D, 0x4E, 0x4F, 0x50, 0x51, \
    0x52, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59, 0x5A, 0x5B, 0x5C,
0x5D, 0x5E, 0x5F \
}

#define EX_SSS_AUTH_NEW_MAC_KEY \
{
    \
    0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, 0x4A, 0x4B,
0x4C, 0x4D, 0x4E, 0x4F, 0x50, 0x51, \
    0x52, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59, 0x5A, 0x5B, 0x5C,
0x5D, 0x5E, 0x5F \
}

#define EX_SSS_AUTH_NEW_DEK_KEY \
{
    \
    0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, 0x4A, 0x4B,
0x4C, 0x4D, 0x4E, 0x4F, 0x50, 0x51, \
    0x52, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59, 0x5A, 0x5B, 0x5C,
0x5D, 0x5E, 0x5F \
}
#endif

```

Note:

We have hard-coded new PlatformSCP03 keys for demonstration purpose. The user can either modify these to their own keys or use a random number generator for randomized keys.

Old Platform SCP03 keys are defined as following. Make sure these match the ones in SE05X.

```

uint8_t OLD_KEY_ENC[AES_MAX_KEY_LEN] = {0};
uint8_t OLD_KEY_MAC[AES_MAX_KEY_LEN] = {0};
uint8_t OLD_KEY_DEK[AES_MAX_KEY_LEN] = EX_SSS_AUTH_SE05X_KEY_DEK;
size_t dek_len = EX_SSS_AUTH_SE05X_KEY_LEN;

```

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, dek_len,  
pCtx);  
ENSURE_OR_GO_CLEANUP(status == kStatus_SSS_Success);
```

Configuring Key Length For Rotation

The ROTATE_PLATFORMSCP03_128BITLEN macro determines the key length during rotation:

- Set to 1 to rotate to 16-byte keys.
- Set to 0 to rotate to 32-byte keys.

By default, it is set to 1, meaning the keys will be rotated to 16 bytes.

5.16.3 Building the Demo

Use the following configurations in CMake:

- SE05X_Auth_PlatformSCP03: ON

Build project: se05x_RotatePlatformSCP03Keys

5.16.4 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.

On Raspberry-Pi or iMX board, run as:

```
./se05x_RotatePlatformSCP03Keys
```

5.16.5 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
```

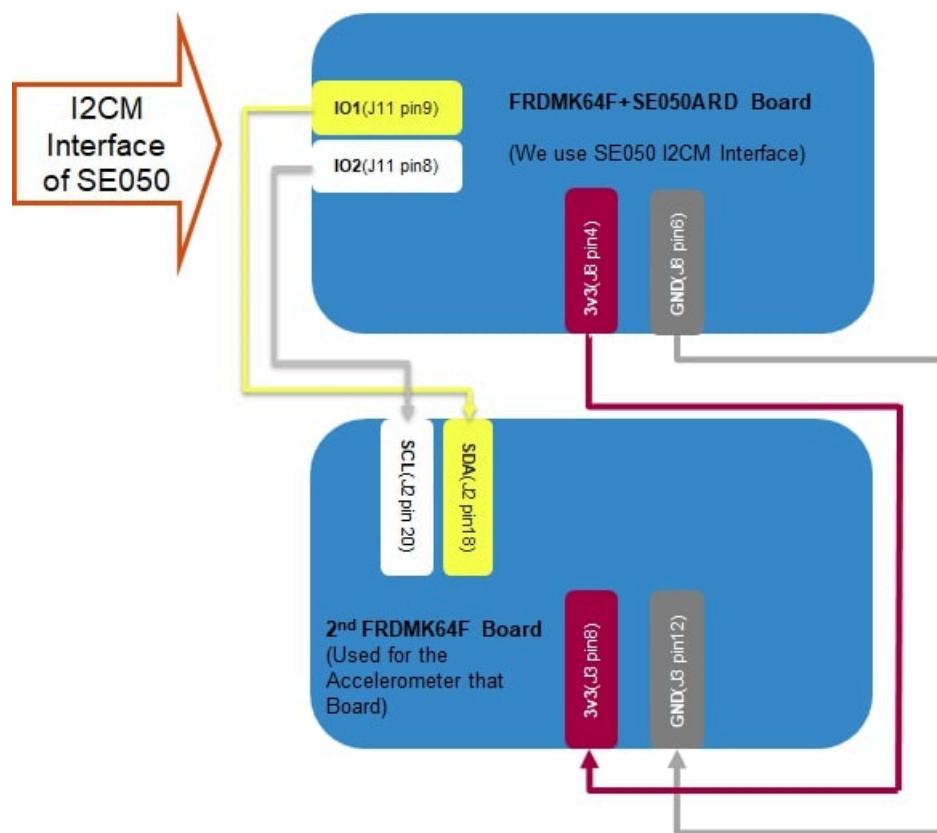
5.17 I2C Master Example

This page is regarding the documentation on I2CM, for more information on I2CM Transaction, See [Section 3.6I2CM / Secure Sensor](#)

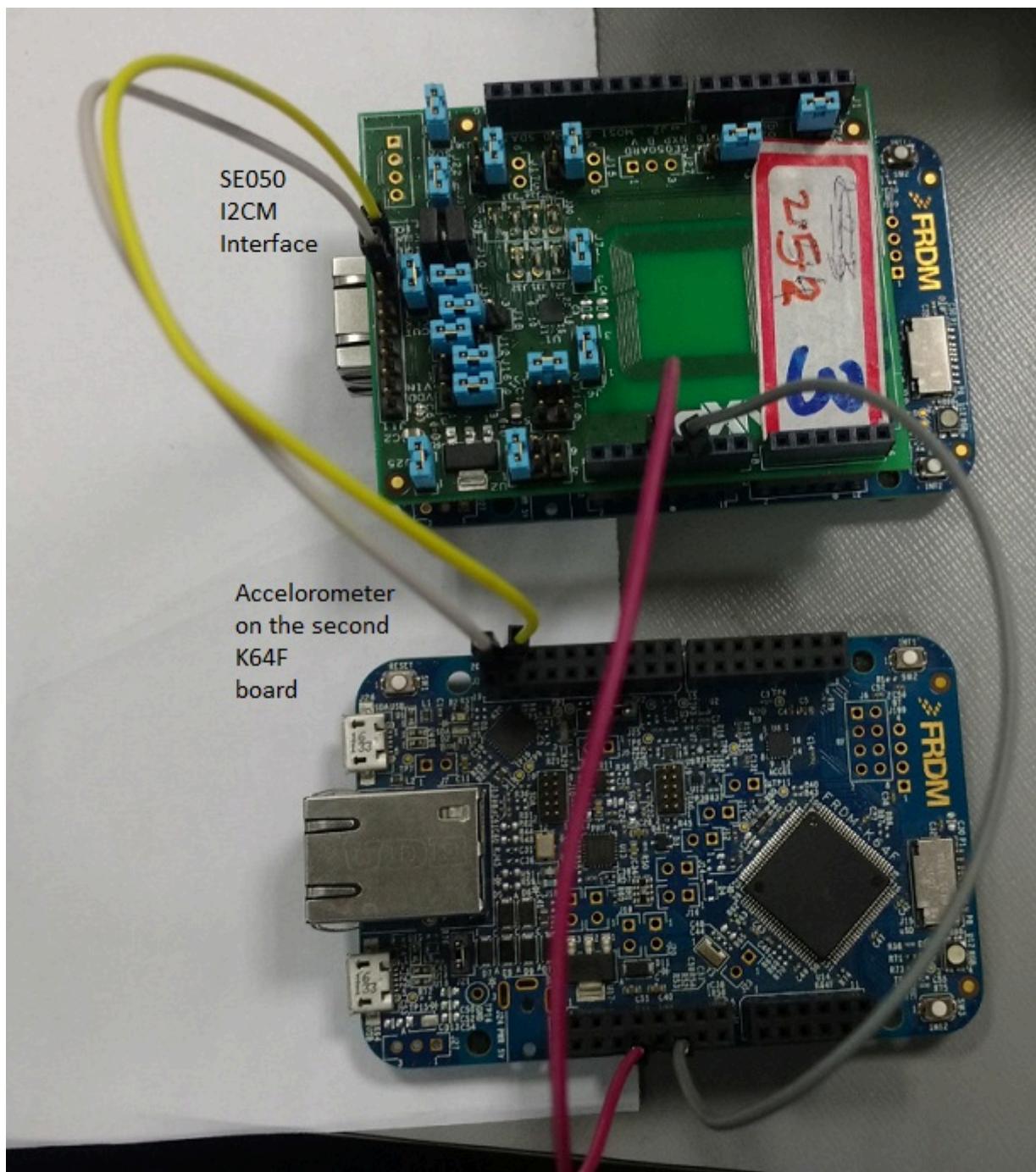
5.17.1 Prerequisites

- 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.

5.17.1.1 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/se05x_I2cMaster` directory to K64F board.
- This binary will put K64F in unoperation state and we will have easy access to Accelerometer through I2C pins.

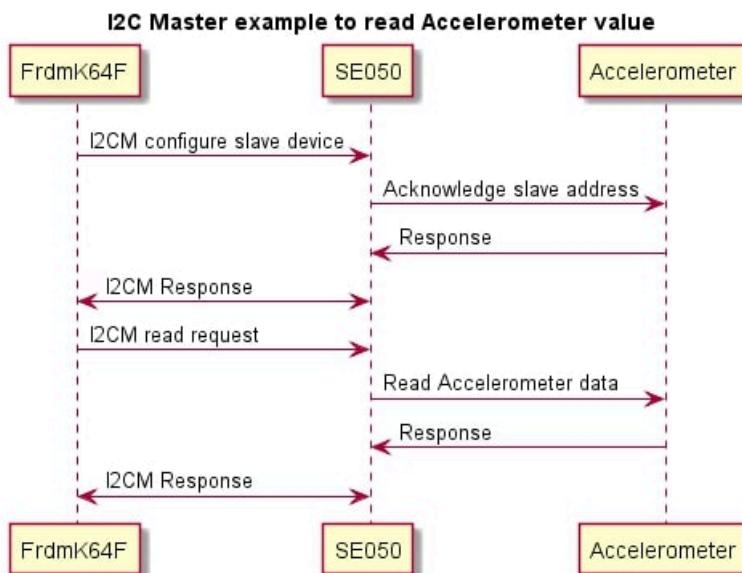
Below is the c code

```
int main(void)
{
    /* Init board hardware. */
    BOARD_InitPins();
    BOARD_BootClockRUN();
    BOARD_InitDebugConsole();%%CRLF
    SP%%__disable_irq();
    while (1)
    {
        __disable_irq();
        __WFI();
    }
}
```

5.17.2 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](#)

5.17.3 Running the Demo

1. Import project `cmake_project_frdmk64f` from `simw-top/projects` directory.
2. Mention `BUILD_TARGET` as `se05x_I2cMaster` or `se05x_I2cMasterWithAttestation` 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 !!!...
```

5.18 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.

5.18.1 Building the Demo

- Build Plug & Trust middleware stack. (Refer [Building / Compiling](#))
- Project: `ex_se05x_WiFiKDF_inject`
- Project: `ex_se05x_WiFiKDF_derive`

5.18.2 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.

5.18.3 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
```

5.19 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 5.7.10SE05X 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.

5.19.1 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.

5.19.2 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_serializedSignedData.bin'.  
App :INFO :ex_sss_export Example Success !?!...  
App :INFO :ex_sss Finished
```

5.20 SE05X Import Transient objects

Before running this example, please run [Section 5.7.9 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.

5.20.1 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.

5.20.2 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
```

5.21 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, 0x4F, 0x31};
/* clang-format on */
```

```

/* API to create buffer */
pse05xWriteBufferSessionCtx->fp_Txn = &se05x_ImportExtObjCreateAPDU;
int index = 0;

sm_status = Se05x_API_WriteSymmKey(pse05xWriteBufferSessionCtx,
    NULL,
    SE05x_MaxAttemps_NA,
    LINE,
    SE05x_KeyID_KEK_NONE,
    keyValue,
    sizeof(keyValue),
    kSE05x_INS_NA,
    kSE05x_SymmKeyType_AES);
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 of the object stores it to the global buffer
 */
smStatus_t se05x_ImportExtObjCreateAPDU(Se05xSession_t *pwrite_apdubufferctx,
    const tlvHeader_t *hdr,
    uint8_t *cmdBuf,
    size_t cmdBufLen,
    uint8_t *rsp,
    size_t *rspLen,
    uint8_t hasle)
{
    AX_UNUSED_ARG(pwrite_apdubufferctx);
    AX_UNUSED_ARG(rsp);
    AX_UNUSED_ARG(rspLen);
    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);
    }
    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;
}
```

```
/* This API transforms buffer to APDU and internally calls transmit and Decrypt
 */
smStatus_t se05x_ImportExtObjTransformBuffer(Se05xSession_t *pSe05xSession,
    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_NOT_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);
        }
    }
```

```

    ENSURE_OR_GO_EXIT(cmdBufLen <= sizeof(gTxBuffer) - i);
    memcpy(&gTxBuffer[i], cmdBuf, cmdBufLen);
    i += cmdBufLen;
}
if (hasle) {
    gTxBuffer[i++] = 0x00;
    gTxBuffer[i++] = 0x00;
}
gTxBufferLen = i;

LOG_D("Here is the buffer of Object needs to be Import");
LOG_AU8_D(gTxBuffer, gTxBufferLen);

if (pSe05xSession->fp_RawTXn) {
    ret = pSe05xSession->fp_RawTXn(
        pgSe05xSessionCtx->conn_ctx, NULL, kSSS_AuthType_None, hdr, cmdBuf,
        cmdBufLen, rsp, rspLen, hasle);
    ENSURE_OR_GO_EXIT(ret == SM_OK);
}
if (pSe05xSession->fp_DeCrypt) {
    ret = pSe05xSession->fp_DeCrypt(pSe05xSession, cmdBufLen, rsp, rspLen,
    hasle);
}
exit:
    return ret;
}

```

```

/* This API transmit the buffer in default session */
static smStatus_t se05x_ImportExtObjTransmitBuffer(void *conn_ctx,
    struct sss_se05x_tunnel_context *pChannelCtx,
    SE_AuthType_t currAuth,
    const tlvHeader_t *hdr,
    uint8_t *cmdBuf,
    size_t cmdBufLen,
    uint8_t *rsp,
    size_t *rspLen,
    uint8_t hasle)
{
    AX_UNUSED_ARG(pChannelCtx);
    AX_UNUSED_ARG(currAuth);
    uint8_t txBuf[SE05X_MAX_BUF_SIZE_CMD] = {0};
    size_t i = 0;
    memcpy(&txBuf[i], hdr, sizeof(*hdr));
    smStatus_t ret = SM_NOT_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) {
            txBuf[i++] = (uint8_t)cmdBufLen;
        }
        else {
            txBuf[i++] = 0x00;
            txBuf[i++] = 0xFFu & (cmdBufLen >> 8);
            txBuf[i++] = 0xFFu & (cmdBufLen);
        }
        memcpy(&txBuf[i], cmdBuf, cmdBufLen);
    }
}

```

```

        i += cmdBufLen;
    }
    else {
        if (cmdBufLen == 0) {
            txBuf[i++] = 0x00;
        }
    }

    if (hasle) {
        txBuf[i++] = 0x00;
        txBuf[i++] = 0x00;
    }

    uint32_t U32rspLen = (uint32_t)*rspLen;
    ret             = (smStatus_t)smCom_TransceiveRaw(conn_ctx, txBuf,
(U16)i, rsp, &U32rspLen);
    *rspLen         = U32rspLen;
    return ret;
}

/* This API decrypts the response buffer */
smStatus_t se05x_ImportExtObjDecryptResponse(
    struct Se05xSession *pSessionCtx, size_t cmd_cmacLen, uint8_t *rsp, size_t
    *rspLength, uint8_t hasle)
{
    U16 rv = SM_NOT_OK;

    if (*rspLength >= 2) {
        rv = rsp[(*rspLength) - 2] << 8 | rsp[(*rspLength) - 1];
        if ((rv == SM_OK) && (pSessionCtx->pdynScp03Ctx != NULL)) {
#if SSS_HAVE_SCP SCP03_SSS
            rv = nxpSCP03_Decrypt_ResponseAPDU(pSessionCtx->pdynScp03Ctx,
cmd_cmacLen, rsp, rspLength, hasle);
#else
            LOG_W("Decrypting without SSS_HAVE_SCP SCP03_SSS");
            rv = SM_NOT_OK;
#endif
        }
#if SSS_HAVE_SCP SCP03_SSS
        else { /*Counter to be increment only in case of authentication is all
kind of SCP
        and response is not 9000 */
            if ((rv != SM_OK) && (pSessionCtx->pdynScp03Ctx != NULL)) {
                if (((pSessionCtx->pdynScp03Ctx->authType ==
kSSS_AuthType_AESKey) ||
                    (pSessionCtx->pdynScp03Ctx->authType ==
kSSS_AuthType_ECKey)) ||
                    ((pSessionCtx->pdynScp03Ctx->authType ==
kSSS_AuthType_SCP03) && (cmd_cmacLen - 8) > 0)) {
                    nxpSCP03_Inc_CommandCounter(pSessionCtx->pdynScp03Ctx);
                }
            }
#endif
        }
    else {
        rv = SM_NOT_OK;
    }
}

```

```
    return rv;
}
```

You can call any of the WriteSecureObject API with your data and create the buffer.

5.21.1 Building

Build the project with the following configurations.

se05x_ImportExternalObjectPrepare

- Project = se05x_ImportExternalObjectPrepare
- SCP=SCP03_SSS
- SE05x_Auth=ECKey

5.21.2 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
<portName>
```

where,

- *keyid* is the authentication keyId at which ECDSA public key is stored.
- *file* is the input ECDSA keypair file (in binary format)
- *portName* is the name of the port over which to connect (COMPORT in case running over VCOM)

5.22 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.

5.22.1 Building the Demo

- Build Plug & Trust middleware stack. (Refer [Building / Compiling](#))
- Project: se05x_MandatePlatformSCP
 - CMake configuration
SE05X_Auth:None

5.22.2 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.

On Raspberry-Pi or iMX board, run as:

```
./se05x_MandatePlatformSCP
```

5.23 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 pre-provisioned EC/RSA keypair as the attestation key and a binary object which will be attested. The pre-provisioned attestation key for Se05x A & C EC keypair is at 0xF0000012 and for Se05x B RSA keypair is at 0xF0000010

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](#)

5.23.1 Building

Build the project with the following configurations.

- Build Plug & Trust middleware stack. (Refer [Building / Compiling](#))
- Project: se05x_ReadWithAttestation

5.23.2 Running

On running the example, you would be able to see object attributes logged on the screen like:

```
App :INFO :Key att data (Len=28)
      00 F5 EF FA    0B 01 00 00    00 00 00 00    00 00 08 00
      00 00 00 00    34 00 00 01    00 00 00 00
App :INFO :Object Id 0xF5EFFA
App :INFO :Object Type 0xB
App :INFO :Type:
App :INFO :     BINARY_FILE
App :INFO :Object Auth Attribute 0x1
App :INFO :Auth:
App :INFO :     Not Set
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 :tagLen for AEAD:0x00
App :INFO :RFU bytes:0x00
App :INFO :Owner:0x0000
App :INFO :Object origin : 0x1
App :INFO :Origin:
App :INFO :     EXTERNAL
App :INFO :Object Version : 0x0000
App :INFO :se05x_ReadWithAttestation 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.

5.23.3 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(sss_se05x_key_store_t *keyStore,
ssss_se05x_object_t *keyObject,
uint8_t *key,
size_t *keylen,
size_t *pKeyBitLen,
ssss_algorithm_t algorithm_attst,
uint8_t *random_attst,
size_t randomLen_attst,
ssss_se05x_attst_data_t *attst_data,
ssss_object_t *verification_object)
{
    AX_UNUSED_ARG(pKeyBitLen);
    AX_UNUSED_ARG(algorithm_attst);
    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 = ATTESTATION_KEY_ID;
    SE05x_AttestationAlgo_t attestAlgo = kSE05x_AttestationAlgo_EC_SHA_256;%CRLF
%SP% /* Variables for verification */
    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 SSS_HAVE_SE05X_VER_GTE_07_02
    uint8_t *pData          = &plainData[0];
    size_t buffLen          = sizeof(plainData);
    uint8_t cmdHashed[32]   = {0};
    size_t cmdHashedLen    = sizeof(cmdHashed);
    int tlvRet              = 0;
    uint8_t timestamp[32]   = {0};
#endif%%CRLF
SP%   status = Se05x_API_ReadSize(&keyStore->session->s_ctx, keyObject->keyId,
    &size);
    ENSURE_OR_GO_CLEANUP(status == SM_OK);%%CRLF
SP%   if (*keylen < size) {
        LOG_E("Insufficient buffer ");
        goto cleanup;
    }%%CRLF
SP%   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;%%CRLF
SP%       signatureLen           = attst_data->data[0].signatureLen;
        attst_data->data[0].timeStampLen = sizeof(SE05x_TimeStamp_t);
#if !SSS_HAVE_SE05X_VER_GTE_07_02
        status = Se05x_API_ReadObject_W_Attst(&keyStore->session->s_ctx,
            keyObject->keyId,
            offset,
            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);
#else%%CRLF
SP%       attst_data->data[0].cmdLen      = sizeof(attst_data->data[0].cmd);
        attst_data->data[0].objSizeLen   = sizeof(attst_data->data[0].objSize);
        attst_data->data[0].attributeLen = sizeof(attst_data->data[0].attribute);
        attst_data->data[0].chipIdLen    = sizeof(attst_data->data[0].chipId);
        pData                          = &plainData[0];
        status                         =
Se05x_API_ReadObject_W_Attst_V2(&keyStore->session->s_ctx,
            keyObject->keyId,
            offset,
            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].chipId,
&(attst_data->data[0].chipIdLen),
attst_data->data[0].cmd,
&(attst_data->data[0].cmdLen),
attst_data->data[0].objSize,
&(attst_data->data[0].objSizeLen),
attst_data->data[0].signature,
&signatureLen);

#endif
if (attst_data->data[0].signatureLen < signatureLen) {
    sss_status = kStatus_SSS_Fail;
    goto cleanup;
}
attst_data->data[0].signatureLen -= signatureLen;
attst_data->valid_number = 1;%%CRLF
SP%%     ENSURE_OR_GO_CLEANUP(status == SM_OK);
//#if !SSS_HAVE_SE05X_VER_GTE_07_02
/* 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
 */%%CRLF
SP%%     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);
#if !SSS_HAVE_SE05X_VER_GTE_07_02
        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;
#else
/* Perform verification operation here on the following data
 *      Perform hash on capdu +
 *      Tag(0x41) + Length + data +
 *      Tag(0x42) + Length + att_data.data[0].chipId +
 *      Tag(0x43) + Length + att_data.data[0].attribute +
```

```
*      Tag(0x44) + Length +dataLen +
*      Tag(0x4F) + Length +att_data.data[0].timestamp
* We perform signature verification on host.
* First we digest the data then pass it to verify API with recv
signature
 * /%CRLF
SP%     sss_status = sss_digest_context_init(
            &digest_ctx, verification_object->keyStore->session,
            digest_algorithm, kMode_SSS_Digest);
        ENSURE_OR_GO_CLEANUP(sss_status == kStatus_SSS_Success);%%CRLF
SP%     LOG_MAU8_I("Cmd ", attst_data->data[0].cmd, attst_data->data[0].cmdLen);%%CRLF
SP%     sss_status = sss_digest_one_go(
            &digest_ctx, attst_data->data[0].cmd, (attst_data->data[0].cmdLen),
            cmdHashed, &cmdHashedLen);
        ENSURE_OR_GO_CLEANUP(sss_status == kStatus_SSS_Success);%%CRLF
SP%     LOG_MAU8_I("Cmd Hashed ", cmdHashed, cmdHashedLen);
        sss_digest_context_free(&digest_ctx);
        plainDataLen = 0;%CRLF
SP%     memcpy(pData, cmdHashed, cmdHashedLen);
        plainDataLen = plainDataLen + cmdHashedLen;
        pData       = pData + plainDataLen;%CRLF
SP%     tlvRet = add_taglength_to_data(&pData, &plainDataLen, buffLen,
kSE05x_TAG_1, (key + offset), max_buffer, true);
        ENSURE_OR_GO_CLEANUP(tlvRet == 0);
        tlvRet = add_taglength_to_data(&pData,
            &plainDataLen,
            buffLen,
            kSE05x_TAG_2,
            attst_data->data[0].chipId,
            attst_data->data[0].chipIdLen,
            true);
        ENSURE_OR_GO_CLEANUP(tlvRet == 0);
        tlvRet = add_taglength_to_data(&pData,
            &plainDataLen,
            buffLen,
            kSE05x_TAG_3,
            attst_data->data[0].attribute,
            attst_data->data[0].attributeLen,
            true);
        ENSURE_OR_GO_CLEANUP(tlvRet == 0);
        tlvRet = add_taglength_to_data(&pData,
            &plainDataLen,
            buffLen,
            kSE05x_TAG_4,
            attst_data->data[0].objSize,
            attst_data->data[0].objSizeLen,
            true);
        ENSURE_OR_GO_CLEANUP(tlvRet == 0);%%CRLF
SP%     memcpy(timestamp, &(attst_data->data[0].timeStamp), attst_data->data[0].timeStampLen);%%CRLF
SP%     tlvRet = add_taglength_to_data(
            &pData, &plainDataLen, buffLen, kSE05x_TAG_TIMESTAMP, timestamp,
            attst_data->data[0].timeStampLen, true);
        ENSURE_OR_GO_CLEANUP(tlvRet == 0);%%CRLF
SP%     LOG_MAU8_I("plainData ", plainData, plainDataLen);

#endif
sss_status = sss_digest_context_init(
```

```
    &digest_ctx, verification_object->keyStore->session,
    digest_algorithm, kMode_SSS_Digest);
    ENSURE_OR_GO_CLEANUP(sss_status == kStatus_SSS_Success);%%CRLF
SP%%        sss_status = sss_digest_one_go(&digest_ctx, plainData, plainDataLen,
    digest, &digestLen);
    ENSURE_OR_GO_CLEANUP(sss_status == kStatus_SSS_Success);%%CRLF
SP%%        sss_digest_context_free(&digest_ctx);%%CRLF
SP%%        /* Verify signature */
        sss_status = sss_asymmetric_context_init(
            &verify_ctx, verification_object->keyStore->session,
            verification_object, algorithm, kMode_SSS_Verify);
    ENSURE_OR_GO_CLEANUP(sss_status == kStatus_SSS_Success);%%CRLF
SP%%        sss_status =
            sss_asymmetric_verify_digest(&verify_ctx, digest, digestLen,
            attst_data->data[0].signature, signatureLen);%%CRLF
SP%%        sss_asymmetric_context_free(&verify_ctx);%%CRLF
SP%%        ENSURE_OR_GO_CLEANUP(sss_status == kStatus_SSS_Success);%%CRLF
SP%%        offset = offset + chunk;
    }

cleanup:
    return sss_status;
}
```

5.24 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 5.7.15SE05X Transport UnLock example](#)

Both the lock and unlock demo use same key and common logic from `se05x_TransportAuth.c`

5.25 SE05X Transport UnLock example

This demo UnLocks SE05x applet, after it is locked by [Section 5.7.14SE05X 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 5.7.14SE05X Transport Lock example](#)

Both the lock and unlock demo use same key and common logic from `se05x_TransportAuth.c`

5.26 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.

5.26.1 Building

Build the project with the following configuration.

- Applet=SE05X_C
- Project = se05x_TimeStamp

5.26.2 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.

5.27 SE05X PCR example

This project can be used to demonstrate PCR as Policy.

5.27.1 Building the Demo

- Build Plug & Trust middleware stack. (Refer [Building / Compiling](#))
- Project: se05x_PCR

5.27.2 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_PCR.exe <PORT NAME>
```

Where `<PORT NAME>` is the VCOM COM port.

On Raspberry-Pi or iMX board, run as:

```
./se05x_PCR
```

5.27.3 Console output

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

```
APDU :DEBUG:CheckObjectExists []
APDU :DEBUG:kSE05x_TAG_1 [object id] = 0xEF000040
```

```
smCom :DEBUG:Tx> (Len=11)
 80 04 00 27 06 41 04 EF 00 00 40
smCom :DEBUG:<Rx (Len=5)
 41 01 02 90 00
sss :DEBUG:sss_key_store_generate_key(@EF000040,
 cipherType=kSSS_CipherType_EC_NIST_P, keyBitLen=256)

APDU :DEBUG:ReadECCurveList []
smCom :DEBUG:Tx> (Len=5)
 80 02 0B 25 00
smCom :DEBUG:<Rx (Len=23)
 41 82 00 11 01 01 02 01 01 01 01 01 01 01 01 01 01
 01 01 01 01 01 90 00

APDU :DEBUG:CheckObjectExists []
APDU :DEBUG:kSE05x_TAG_1 [object id] = 0xEF000040
smCom :DEBUG:Tx> (Len=11)
 80 04 00 27 06 41 04 EF 00 00 40
smCom :DEBUG:<Rx (Len=5)
 41 01 02 90 00

APDU :DEBUG:WriteECKey []
APDU :DEBUG:kSE05x_TAG_POLICY
APDU :DEBUG:policy (Len=45)
 2C 00 00 00 00 1F FD 90 00 7F FF FF FF 87 D3 E3
 93 19 8F 5C 80 E0 BC 9B C9 82 00 1F B0 EE 20 1C
 27 0B 6D C8 84 52 E4 13 A3 25 56 81 75
APDU :DEBUG:kSE05x_TAG_MAX_ATTEMPTS [maxAttempt] = 0x0
APDU :DEBUG:kSE05x_TAG_1 [object id] = 0xEF000040
APDU :DEBUG:kSE05x_TAG_2 [curveID] = 0x3
APDU :DEBUG:kSE05x_TAG_3 [privKey] (Len=0)
APDU :DEBUG:kSE05x_TAG_4 [pubKey] (Len=0)
smCom :DEBUG:Tx> (Len=61)
 80 01 61 00 38 11 2D 2C 00 00 00 00 1F FD 90 00
 7F FF FF FF 87 D3 E3 93 19 8F 5C 80 E0 BC 9B C9
 82 00 1F B0 EE 20 1C 27 0B 6D C8 84 52 E4 13 A3
 25 56 81 75 41 04 EF 00 00 40 42 01 03
smCom :DEBUG:<Rx (Len=2)
 90 00

APDU :DEBUG:ECDSASign []
APDU :DEBUG:kSE05x_TAG_1 [objectID] = 0xEF000040
APDU :DEBUG:kSE05x_TAG_2 [ecSignAlgo] = 0x21
APDU :DEBUG:kSE05x_TAG_3 [inputData] (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
smCom :DEBUG:Tx> (Len=48)
 80 03 0C 09 2B 41 04 EF 00 00 40 42 01 21 43 20
 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
smCom :DEBUG:<Rx (Len=2)
 69 85
sss :WARN :nxEnsure:'ret == SM_OK' failed. At Line:5601 Function:sss_se05x_Tx
App :INFO :Sign failed due to PCR expected value mismatch!!!

APDU :DEBUG:CheckObjectExists []
APDU :DEBUG:kSE05x_TAG_1 [object id] = 0x7FFFFFFF
smCom :DEBUG:Tx> (Len=11)
 80 04 00 27 06 41 04 7F FF FF FF
smCom :DEBUG:<Rx (Len=5)
```

```

41 01 02 90      00

APDU :DEBUG:WritePCR []
APDU :INFO :Policy is NULL
APDU :DEBUG:kSE05x_TAG_1 [object id] = 0x7FFFFFFF
APDU :DEBUG:kSE05x_TAG_2 [initialValue] (Len=32)
12 A1 49 82      32 00 00 00      00 00 00 00      00 00 00 00
00 00 00 00      00 00 00 00      00 00 00 00      00 00 00 00
APDU :DEBUG:kSE05x_TAG_3 [inputData] (Len=0)
smCom :DEBUG:Tx> (Len=45)
80 01 09 00      28 41 04 7F      FF FF FF 42      20 12 A1 49
82 32 00 00      00 00 00 00      00 00 00 00      00 00 00 00
00 00 00 00      00 00 00 00      00 00 00 00      00
smCom :DEBUG:<Rx (Len=2)
90 00

APDU :DEBUG:WritePCR []
APDU :INFO :Policy is NULL
APDU :DEBUG:kSE05x_TAG_1 [object id] = 0x7FFFFFFF
APDU :DEBUG:kSE05x_TAG_2 [initialValue] (Len=0)
APDU :DEBUG:kSE05x_TAG_3 [inputData] (Len=5)
12 A1 49 82      32
smCom :DEBUG:Tx> (Len=18)
80 01 09 00      0D 41 04 7F      FF FF FF 43      05 12 A1 49
82 32
smCom :DEBUG:<Rx (Len=2)
90 00

APDU :DEBUG:ECDSASign []
APDU :DEBUG:kSE05x_TAG_1 [objectId] = 0xEF000040
APDU :DEBUG:kSE05x_TAG_2 [ecSignAlgo] = 0x21
APDU :DEBUG:kSE05x_TAG_3 [inputData] (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
smCom :DEBUG:Tx> (Len=48)
80 03 0C 09      2B 41 04 EF      00 00 40 42      01 21 43 20
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
smCom :DEBUG:<Rx (Len=76)
41 82 00 46      30 44 02 20      50 06 84 FF      D1 CE 87 E7
F6 89 BE 6C      5B 93 EE EB      26 FB 9D 86      01 25 4F C5
CE EF 4E 0D      68 9A 64 D9      02 20 6A 4F      27 D0 95 CC
BD D4 20 66      60 B2 9E 52      26 96 A1 31      3C 1F 7D 77
E3 16 02 AA      B2 E3 36 61      05 70 90 00

APDU :DEBUG:ECDSAVerify []
APDU :DEBUG:kSE05x_TAG_1 [objectId] = 0xEF000040
APDU :DEBUG:kSE05x_TAG_2 [ecSignAlgo] = 0x21
APDU :DEBUG:kSE05x_TAG_3 [inputData] (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
APDU :DEBUG:kSE05x_TAG_5 [signature] (Len=70)
30 44 02 20      50 06 84 FF      D1 CE 87 E7      F6 89 BE 6C
5B 93 EE EB      26 FB 9D 86      01 25 4F C5      CE EF 4E 0D
68 9A 64 D9      02 20 6A 4F      27 D0 95 CC      BD D4 20 66
60 B2 9E 52      26 96 A1 31      3C 1F 7D 77      E3 16 02 AA
B2 E3 36 61      05 70
smCom :DEBUG:Tx> (Len=120)
80 03 0C 0A      73 41 04 EF      00 00 40 42      01 21 43 20
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
45 46 30 44      02 20 50 06      84 FF D1 CE      87 E7 F6 89
BE 6C 5B 93      EE EB 26 FB      9D 86 01 25      4F C5 CE EF
4E 0D 68 9A      64 D9 02 20      6A 4F 27 D0      95 CC BD D4
20 66 60 B2      9E 52 26 96      A1 31 3C 1F      7D 77 E3 16
02 AA B2 E3      36 61 05 70

smCom :DEBUG:<Rx (Len=5)
41 01 01 90 00
App :INFO :Sign & Verify is Success with Expected PCR value!!!
APDU :DEBUG:DeleteSecureObject []
APDU :DEBUG:kSE05x_TAG_1 [object id] = 0xEF000040
smCom :DEBUG:Tx> (Len=11)
80 04 00 28 06 41 04 EF 00 00 40
smCom :DEBUG:<Rx (Len=2)
90 00

APDU :DEBUG:DeleteSecureObject []
APDU :DEBUG:kSE05x_TAG_1 [object id] = 0x7FFFFFFF
smCom :DEBUG:Tx> (Len=11)
80 04 00 28 06 41 04 7F FF FF FF
smCom :DEBUG:<Rx (Len=2)
90 00
App :INFO :ex_sss Finished
Press any key to continue . . .
```

5.28 Configuring Applet Features

This tool is used to configure the applet for different variants, i.e. Variant A, Variant B and Variant C. Note that this tool only enables or disables the features according to the configured variant, it **does not alter the variant itself**. The tool should be compiled with ECKey encryption, and optionally PlatfSCP03 encryption also. You would need to provide a command line parameter for ECKey key to be used, and an environment variable for your own Platform SCP03 keys, if required.

5.28.1 Configuring ECKey key for command line

This example takes as a command-line input, the filename in which ECDSA key is stored. The key should be stored in `.der` (binary) format only. If the available key is in `.pem` format, it can be converted into binary format using OpenSSL command-line utility. Refer to [OpenSSL Commands](#) for more information on how to convert `.pem` formatted file to `.der` formatted file.

5.28.2 Configuring Environment for PlatfSCP03 Keys

If you do not wish to use Platform SCP03 encryption, skip to the next step.

To use your own platform SCP03 keys, refer to [Section 10.11Using own Platform SCP03 Keys](#) on details on setting up your environment.

5.28.3 Using with PC as host

1. Flash the VCOM binary present in `binaries` directory on to the target board.
2. Note the VCOM `COMPORT` from device manager.
3. Build the project `se05x_setAppletFeatures`. Configure the tool with `SE050_A`, `SE050_B` or `SE050_C`, to enable the applet specific features.

4. Run the executable for desired variant as:

```
cd tools
set EX_SSS_BOOT SCP03_PATH_ENV=\path\to\platfscp03\keys
se05x_setAppletFeatures.exe -file <filename> <COMPORT>
```

Where EX_SSS_BOOT SCP03_PATH_ENV is set to the path of file containing Platform SCP03 keys as described in [Configuring Environment for PlatfSCP03 Keys](#).

COMPORT is the port obtained from step 2.

filename is the path to file containing ECDSA keypair provisioned at **RESERVED_ID_FEATURE** in binary format as described in [Configuring ECKey key for command line](#)

5.28.4 Using with iMX-6 as host

1. Build the project se05x_setAppletFeatures. Configure the tool with SE050_A, SE050_B or SE050_C, to enable the applet specific features.
2. Run the executable for desired variant as:

```
cd tools
export EX_SSS_BOOT SCP03_PATH_ENV=\path\to\platfscp03\keys
./se05x_setAppletFeatures -file <filename>
```

Where EX_SSS_BOOT SCP03_PATH_ENV is set to the path of file containing Platform SCP03 keys as described in [Configuring Environment for PlatfSCP03 Keys](#).

filename is the path to file containing ECDSA keypair provisioned at **RESERVED_ID_FEATURE** in binary format as described in [Configuring ECKey key for command line](#)

5.29 Inject Certificate into SE

This tool is to demonstrate how to inject certificates into the SE. The keyIDs where certificates are injected are defined in se05x.InjectCertificate.h:

```
#define ECC_CERTIFICATE_KEYID 0x000000014
#define RSA_CERTIFICATE_KEYID 0x000000011
```

Note:

Certificates to be injected are also defined in se05x.InjectCertificate.h file. These can be changed as required.

5.29.1 Building the example

Use the following CMake configurations to compile the example

- Project: se05x.InjectCertificate

5.29.2 How to use

Run the tool as:

```
se05x.InjectCertificate <port>
```

- For systems connecting with T=1 over I2C, the port parameter can be skipped.

5.30 SE05X Read State example

This project can be used to get SE05X state information. The state information is w.r.t

1. LockState
2. Restriction on Object Creation
3. Platform SCP mandate state

Refer to [SE05x_LockState_t](#) Refer to [SE05x_RestrictMode_t](#) Refer to
[SE05x_PlatformSCPRequest_t](#)

```
SE05x_LockState_t lockState;
SE05x_RestrictMode_t restrictMode;
SE05x_PlatformSCPRequest_t platformSCPRequest;
```

5.30.1 Building the Demo

- Build Plug & Trust middleware stack. (Refer [Building / Compiling](#))
- Project: se05x_ReadState

5.30.2 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_ReadState.exe <PORT NAME>
```

Where **<PORT NAME>** is the VCOM COM port.

On Raspberry-Pi or iMX board, run as:

```
./se05x_ReadState
```

5.30.3 Console output

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

```
APDU :DEBUG:ReadState []
smCom :DEBUG:Tx> (Len=5)
 80 02 00 5B 00
smCom :DEBUG:<Rx (Len=9)
 41 82 00 03 02 00 02 90 00
App :INFO :SE05x Read State Successfully!!!
App :INFO :Following is the SE05x Read State status
App :INFO :SE05x Lock State = 0x2 i.e. SE05x is Unlocked!!!
App :INFO :SE05x Restrict Mode = 0x0 i.e. No Restriction is applied for
 object creation!!!
App :INFO :SE05x Platform SCP Request = 0x2 i.e. Platform SCP is not required
 for Communication!!!
App :INFO :ex_sss Finished
Press any key to continue . . .
```

5.31 SE05X Personalization Remove RSA Key Generation Module

This project permanently deletes rsa key generation module from secure element. Running this example will restrict the generation of rsa keys inside the secure element.

5.31.1 Building the Demo

- Build Plug & Trust middleware stack. (Refer [Building / Compiling](#))
- Project: se05x_Perso_Delete_Mod_RSAKeyGen

5.31.2 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_Perso_Delete_Mod_RSAKeyGen.exe <PORT NAME>
```

Where **<PORT NAME>** is the VCOM COM port.

On Raspberry-Pi or iMX board, run as:

```
./se05x_Perso_Delete_Mod_RSAKeyGen
```

5.31.3 Console output

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

```
App :INFO :# Running Remove Module RSAKEYGEN : On chip generation of RSA keys.
App :INFO :# Remove Module RSAKEYGEN is complete.
App :INFO :ex_sss Finished
```

5.32 DEMO for Personalization of SE051

This demo can be used to personalize some of the parameters for SE051.

The demo is self explanatory:

```
se05x_Personalization.exe <operation> [operation] <port_name>

Where, 'operation': one or sequence of the following:%%CRLF
SP%%                               HELP : Prints list of APIs
I2C_DisableClockStretching : I2C: Disables clock stretching
I2C_EnableClockStretching : I2C: Enables clock stretching
    I2C_ProtocolGP : I2C: Use GP Mode for I2C
    I2C_ProtocolUM : I2C: Use UM Mode like SE050 for I2C
I2C_ProtocolAutoDetect : I2C: Automatically detect I2C Protocol
    I2C_ProtocolManual : I2C: Manually use fixed Protocol
    I2C_EnablePowerSave : I2C: Go to power save after last APDU
    I2C_DisablePowerSave : I2C: DO not go to power save after last APDU
        I2C_SemiBlocking : I2C: SEMI Blocking I2C
        I2C_NonBlocking : I2C: Non Blocking I2C
    RM_MOD_IOT_EXTENDED : Delete: Extension to Elliptic Curve
Cryptography.
        RM_MOD_IOT_BASE : Delete: Collection of crypto algorithms often
used in IoT devices.
    RM_MODX_EGOVACCELERATORS : Delete: Secure Messaging Accelerators for
eGovernment applications and modular arithmetic math API.
        RM_MOD_RSAKEYGEN : On chip generation of RSA keys.
```

For further details, please see product user manual.

Note:

You can add more operations in a single command.

Warning:

These values are effective only after IC Reset.

5.32.1 Compatible settings from host and SE

Some parameters need compatible settings at both sides, from host and SE. Else it leaves system in a non-operable state.

e.g. If the SE is configured to be in ProtocolGP, and host is using ProtocolUM, there can't be any further communication with the SE without re-compiling the middleware/demo at the host. And since the communication between Host and SE is not function, there is no direct/easy way to re-configure the SE Back.

5.33 SE05X MultiThread demo

This project demonstrates two threads, opening two sessions with the se05x using tunneling api's. Each thread performing ecc sign and verify operations.

If we use user sessions then at least two authentication objects need to be provisioned in se05x using the delete and test provision utility before running the demo:

- For an overview of authentication objects refer to [Auth Objects](#)
- For a provisioning example refer to [Delete and Test Provision](#)

In Multi-step operations the concurrent execution with same cipher mode is not allowed. (Refer [Known limitations](#))

Note:

This demo can be build for FreeRTOS on embedded platforms or for pthread on POSIX systems (e.g. Linux).

5.33.1 Prerequisites

- Micro USB cable
- FRDM-K64F/imx-RT1060/Raspberry-Pi/iMX board
- Personal Computer
- SE05x Arduino shield

5.33.2 Building the Demo

- Build Plug & Trust middleware stack. (Refer [Building / Compiling](#))
- Project: `se05x_MultiThread`

5.33.3 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_Multithread.exe <PORT NAME>
```

Where `<PORT NAME>` is the VCOM COM port.

On Raspberry-Pi or iMX board, run as:

```
./se05x_Multithread
```

5.33.4 Console output

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

```
sss :WARN :Communication channel is with Plain.  
sss :WARN :!!!Not recommended for production use.!!!  
sss :WARN :Communication channel is with Plain.  
sss :WARN :!!!Not recommended for production use.!!!  
App :INFO :Key Store Generate Key success user 2!!!  
App :INFO :Key Store Generate Key success user 1!!!  
App :INFO :sss_asymmetric_sign_digest success user 2  
App :INFO :sss_asymmetric_sign_digest success user 1  
App :INFO :Set public key success user 2  
App :INFO :Set public key success user 1  
App :INFO :sss_asymmetric_verify_digest success user 2  
App :INFO :Done User 2  
App :INFO :sss_asymmetric_verify_digest success user 1  
App :INFO :Done User 1  
App :INFO :SE05x Multithreaded example finished
```

5.34 SE05X Invoke Garbage Collection Example

The example will trigger the garbage collection in SE05X. Invoking `CM_InvokeGarbageCollection` api will close the existing session to SE05X. To use SE05X, open session again.

Warning:

Excessive calls to `CM_InvokeGarbageCollection` API will impact the device durability

5.34.1 Building the Demo

- Build Plug & Trust middleware stack. (Refer [Building / Compiling](#))
- Project: `se05x_InvokeGarbageCollection`

5.34.2 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_InvokeGarbageCollection.exe <PORT NAME>
```

Where `<PORT NAME>` is the VCOM COM port.

On Raspberry-Pi or iMX board, run as:

```
./se05x_InvokeGarbageCollection
```

5.34.3 Console output

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

```
hostLib:WARN :Invoking this api will close the existing session to SE05X.  
hostLib:WARN :To use SE05X, open session again.  
hostLib:INFO :Sending Select command  
hostLib:INFO :Command Successful!!!  
hostLib:INFO :Command Response (Len=18)  
    6F 10 84 08      A0 00 00 01      51 00 00 00      A5 04 9F 65  
    01 FF  
hostLib:INFO :Sending 'Invoke Garbage Collection' Command  
hostLib:INFO :Command Successful!!!  
hostLib:INFO :Command Response (Len=5)  
    FE 03 DF 25      00  
App   :INFO :se05x_InvokeGarbageCollection Success !!!...  
App   :INFO :ex_sss Finished
```

5.35 ECC Concurrent Example

This project demonstrates Elliptic Curve Cryptography sign and verify operations using SSS APIs. On a multiprocess OS it can be run as multiple instances. Before running an instance ensure the secure element is properly provisioned. The utility [Delete and Test Provision](#) can be used to provision the authentication objects used by this example. Invoke `se05x_Delete_and_test_provision` once during the preparation of the demo. The authentication objects are required for user sessions of type UserID/AESKey/ECKey.

5.35.1 Building the Demo

- Build Plug & Trust middleware stack. (Refer [Building / Compiling](#))
- When used in combination with Access Manager compile for SMCOM = JRCPv1

5.35.2 Restrictions

- Each user session needs to have a different authentication object

5.35.3 Running the Example in combination with the Access Manager

For additional information on the AccessManager refer to [Access Manager: Manage access from multiple \(Linux\) processes to an SE05x IoT Applet](#).

5.35.4 Usage

The client process opens an authenticated user session (None/UserID/AESkey/ECKey) configured at build time. The user session must be without platform SCP03. Platform SCP03 is handled by the Access Manager.

The client process connects to the AccessManager over JRCPv1.

Two concurrent client processes (establishing a user session) can connect to the AccessManager.

This program takes 4 command line arguments:

- authid: Authentication object id with which the session can be opened. It can be of type UserID/AESkey/ECKey (or authentication type 'None'). In case the user session authentication is of type 'None' pass '0' as argument
- keyid: The id at which the functional Ec key pair will be stored, use a unique value per process.
- count (optional argument): Number of times the Ec Sign Verify operation will be repeated

- port: Connection port to AccessManager. e.g. 127.0.0.1:8040

An example invocation of the program (passing the Authentication Id of a UserID object) is:

```
./se05x_ConcurrentEcc -authid 0x7DA00001 -keyid 0xEF001234 -cnt 100 -port
127.0.0.1:8040
```

5.35.5 Concurrent Usage

The following command invocations illustrate how two processes connect to the AccessManger using different Authentication Id's and (functional) key Id's.

The following Authentication Id's are provisioned in SE using the “delete & provision” utility

AuthId	1	2
UserID	0x7DA00001	0x7DA00011
AESkey	0x7DA00002	0x7DA00012
ECKey	0x7DA00003	0x7DA00013

KeyId can be anything above 0xEF000000, choose a different value for each client process.

- Ec Sign / Verify Auth=ID_1 Auth=ID_2
 - ./se05x_ConcurrentEcc -authid 0x7DA00001 -keyid 0xEF001234 -cnt 100 -port 127.0.0.1:8040
 - ./se05x_ConcurrentEcc -authid 0x7DA00011 -keyid 0xEF001244 -cnt 100 -port 127.0.0.1:8040
- Ec Sign / Verify Auth=None Auth=EC
 - ./se05x_ConcurrentEcc -authid 0 -keyid 0xEF001236 -cnt 100 -port 127.0.0.1:8040
 - ./se05x_ConcurrentEcc -authid 0x7DA00003 -keyid 0xEF001246 -cnt 100 -port 127.0.0.1:8040
- Ec Sign / Verify Auth=AES_1 Auth=AES_2
 - ./se05x_ConcurrentEcc -authid 0x7DA00002 -keyid 0xEF001238 -cnt 100 -port 127.0.0.1:8040
 - ./se05x_ConcurrentEcc -authid 0x7DA00012 -keyid 0xEF001248 -cnt 100 -port 127.0.0.1:8040

5.36 Symmetric Multi Step Concurrent Example

This project demonstrates Symmetric Multi step operations. using SSS APIs. On a multiprocess OS it can be run as multiple instances. Before running an instance ensure the secure element.is properly provisioned. The block cipher mode to apply (ECB or CBC) can be chosen by the user. The utility [Delete and Test Provision](#) can be used to provision the authentication objects used by this example. Invoke `se05x_Delete_and_test_provision` once during the preparation of the demo. The authentication objects are required for user sessions of type UserID/AESKey/ECKey.

5.36.1 Building the Demo

- Build Plug & Trust middleware stack. (Refer [Building / Compiling](#))
- When used in combination with Access Manager compile for SMCOM = JRCPv1

5.36.2 Restrictions

- Each user session needs to have a different authentication object
- The same block cipher mode cannot be used for two concurrent client processes. If one process uses ECB the other must use CBC. This is a restriction from the SSS layer.

5.36.3 Running the Example in combination with the Access Manager

For additional information on the AccessManager refer to [Access Manager: Manage access from multiple \(Linux\) processes to an SE05x IoT Applet](#).

5.36.4 Usage

The client process opens an authenticated user session (None/UserID/AESkey/ECkey) configured at build time. The user session must be without platform SCP03. Platform SCP03 is handled by the Access Manager.

The client process connects to the AccessManager over JRCPv1.

Two concurrent client processes (establishing a user session) can connect to the AccessManager.

This program takes 5 command line arguments:

- authid: Authentication object id with which the session can be opened. It can be of type UserID/AESkey/ECkey (or authentication type 'None'). In case the user session authentication is of type 'None' pass '0' as argument
- keyid: The id at which the symmetric key has to be stored, use a unique value per process.
- algo: Block cipher mode, use 1 for ECB and 2. for CBC
- count (optional argument): Number of times the crypto operation will be repeated.
- port: Connection port to AccessManager. e.g. 127.0.0.1:8040

An example invocation of the program is:

```
./se05x_ConcurrentSymm -authid 0x7DA00001 -keyid 0xEF002234 -algo 1 -cnt 100 -port 127.0.0.1:8040
```

5.36.5 Concurrent Usage

The following command invocations illustrate how two processes connect to the AccessManger using different Authentication Id's and (functional) key Id's.

The following Authentication Id's are provisioned in SE using the "delete & provision" utility

AuthId	1	2
UserID	0x7DA00001	0x7DA00011
AESkey	0x7DA00002	0x7DA00012
ECKey	0x7DA00003	0x7DA00013

KeyId can be anything above 0xEF000000, choose a different value for each client process.

- ECB Enc/Dec CBC Enc/Dec Auth=ID Auth=AES
 - ./se05x_ConcurrentSymm -authid 0x7DA00001 -keyid 0xEF002234 -algo 1 -cnt 100 -port 127.0.0.1:8040
 - ./se05x_ConcurrentSymm -authid 0x7DA00002 -keyid 0xEF002235 -algo 2 -cnt 100 -port 127.0.0.1:8040
- ECB Enc/Dec CBC Enc/Dec Auth=ID_1 Auth=ID_2
 - ./se05x_ConcurrentSymm -authid 0x7DA00001 -keyid 0xEF002236 -algo 1 -cnt 100 -port 127.0.0.1:8040
 - ./se05x_ConcurrentSymm -authid 0x7DA00011 -keyid 0xEF002237 -algo 2 -cnt 100 -port 127.0.0.1:8040
- ECB Enc/Dec; CBC Enc/Dec Auth=EC Auth=AES
 - ./se05x_ConcurrentSymm -authid 0x7DA00003 -keyid 0xEF002238 -algo 1 -cnt 100 -port 127.0.0.1:8040
 - ./se05x_ConcurrentSymm -authid 0x7DA00002 -keyid 0xEF002239 -algo 2 -cnt 100 -port 127.0.0.1:8040

5.37 Delete and Test Provision

'Delete and Test Provision' is a utility that deletes a subset of the crypto objects stored on the secure element. Next it provisions different types of key objects (as documented in the table below). These key objects enable various use cases with the secure element. As an example: the authentication objects for session based authentication (UserID/AESkey/ECKey) are provisioned through this utility.

To enable concurrent (up to the maximum of two) authenticated user sessions, two authentication objects of each type (UserID/AESkey/ECKey) are provisioned.

Note:

Authentication objects - of the same object type - are provisioned on IDs with an offset of +0x10.

Warning:

This utility provisions the same value for both authentication object type instances. In a product deployment different values must be used, do not re-use any of the values provisioned by this test utility in a product deployment.

5.37.1 Object IDs provisioned

Object ID	Object Type	Usage
0x7DA0000 1	User ID	Allows the user to Open User ID Auth Session to SE.
0x7DA0000 2	Symm Key	Allows the user to Open AES Key Auth Session to SE.
0x7DA0000 3	EC Key	Allows the user to Open EC Key Auth Session to SE.
0x7DA0001 1	User ID	Allows the user to Open additional User ID Auth Session to SE.
0x7DA0001 2	Symm Key	Allows the user to Open additional AES Key Auth Session to SE.
0x7DA0001 3	EC Key	Allows the user to Open additional EC Key Auth Session to SE.
0x7FFF020 0	Symm Key	Allows the user to switch transport LockState of the SE.
0x7FFF020 1	EC Key	Provisions ECKA pair at SE for EC key Session Authentication.
0x7FFF020 2	EC Key	Provisions ECKA pair at SE for EC key Session Authentication.
0x7FFF020 3	EC Key	Provisions ECKA pair at SE for EC key Session Authentication.
0x7FFF020 4	EC Key	Used for applet personalization.
0x7FFF020 5	User ID	Allows the user to delete all objects. Except those provisioned by NXP.
0x7FFF020 6	Binary	Holds the device unique ID.
0x7FFF020 7	User ID	Allows the user to make platform SCP mandatory or not.

Warning:

Some of the object Ids here are provisioned with same values. This is for test, example and demo purpose only.

5.37.2 Authentication Keys**Warning:**

*These values are just for demonstration. The user **MUST** modify these values in the secure element and the application for real world use cases.*

User ID

```
#define EX_SSS_AUTH_SE05X_UserID_AUTH_ID kEX_SSS_ObjID_UserID_Auth
#define EX_SSS_AUTH_SE05X_UserID_VALUE \
{ \
    0xC0, 0x01, 0x02, 0x03, 0x04 \
} /* COOL 234*/
#define EX_SSS_AUTH_SE05X_UserID_VALUE2 \
{ \
    0xC0, 0x01, 0x02, 0x03, 0x04, 0x05 \
} /* COOL 2345*/
```

Applet SCP

```
#define EX_SSS_AUTH_SE05X_APPLETSCP_AUTH_ID kEX_SSS_ObjID_APPLETSCP03_Auth
#define EX_SSS_AUTH_SE05X_APPLETSCP_VALUE \
{ \
    0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, 0x4A, \
    0x4B, 0x4C, 0x4D, 0x4E, 0x4F \
}
#define EX_SSS_AUTH_SE05X_APPLETSCP_VALUE2 \
{ \
    0xea, 0x62, 0x04, 0x48, 0x0b, 0xf5, 0x19, 0xf6, 0xc2, 0xb7, 0x7f, \
    0xba, 0x8b, 0x2d, 0x57, 0x30 \
}
```

EC Key

```
#define EX_SSS_AUTH_SE05X_ECKEY_ECDSA_AUTH_ID kEX_SSS_ObjID_ECKEY_Auth
#define EX_SSS_AUTH_SE05X_KEY_HOST_ECDSA_KEY \
{ \
    0x30, 0x81, 0x87, 0x02, 0x01, 0x00, 0x30, 0x13, \
    0x06, 0x07, 0x2A, 0x86, 0x48, 0xCE, 0x3D, 0x02, \
    0x01, 0x06, 0x08, 0x2A, 0x86, 0x48, 0xCE, 0x3D, \
    0x03, 0x01, 0x07, 0x04, 0x6D, 0x30, 0x6B, 0x02, \
    0x01, 0x01, 0x04, 0x20, \
    0x6D, 0x2F, 0x43, 0x2F, 0x8A, 0x2F, 0x45, 0xEC, \
    0xD5, 0x82, 0x84, 0x7E, 0xC0, 0x83, 0xBB, 0xEB, \
    0xC2, 0x3F, 0x1D, 0xF4, 0xF0, 0xDD, 0x2A, 0x6F, \
    0xB8, 0x1A, 0x24, 0xE7, 0xB6, 0xD5, 0x4C, 0x7F, \
    0xA1, 0x44, 0x03, 0x42, 0x00, \
    0x04, 0x3C, 0x9E, 0x47, 0xED, 0xF0, 0x51, 0xA3, \
    0x58, 0x9F, 0x67, 0x30, 0x2D, 0x22, 0x56, 0x7C, \
    0x2E, 0x17, 0x22, 0x9E, 0x88, 0x83, 0x33, 0x8E, \
}
```

```

        0xC3, 0xB7, 0xD5, 0x27, 0xF9, 0xEE, 0x71, 0xD0, \
        0xA8, 0x1A, 0xAE, 0x7F, 0xE2, 0x1C, 0xAA, 0x66, \
        0x77, 0x78, 0x3A, 0xA8, 0x8D, 0xA6, 0xD6, 0xA8, \
        0xAD, 0x5E, 0xC5, 0x3B, 0x10, 0xBC, 0x0B, 0x11, \
        0x09, 0x44, 0x82, 0xF0, 0x4D, 0x24, 0xB5, 0xBE, \
        0xC4
    }

#define EX_SSS_AUTH_SE05X_KEY_HOST_ECDSA_KEY2 \
{
    0x30, 0x81, 0x87, 0x02, 0x01, 0x00, 0x30, 0x13, \
    0x06, 0x07, 0x2A, 0x86, 0x48, 0xCE, 0x3D, 0x02, \
    0x01, 0x06, 0x08, 0x2A, 0x86, 0x48, 0xCE, 0x3D, \
    0x03, 0x01, 0x07, 0x04, 0x6D, 0x30, 0x6B, 0x02, \
    0x01, 0x01, 0x04, 0x20, \
    0x12, 0xe2, 0xd3, 0xc7, 0x31, 0xa6, 0x7c, 0x32, \
    0xfb, 0xd7, 0x2f, 0xa9, 0xc4, 0xbb, 0xc2, 0xd0, \
    0x64, 0xad, 0x50, 0x99, 0xd3, 0x3d, 0x01, 0x4b, \
    0x4f, 0x36, 0x90, 0x9c, 0xba, 0xab, 0xbb, 0xda, \
    0xA1, 0x44, 0x03, 0x42, 0x00, \
    0x04, 0x0d, 0x0e, 0x03, 0xdd, 0x40, 0xle, 0x77, \
    0xff, 0xab, 0xa8, 0xb5, 0x79, 0xdb, 0x8a, 0xf4, \
    0x09, 0x7b, 0x59, 0x4e, 0xe8, 0xa0, 0xb8, 0x1c, \
    0xeb, 0xa8, 0x53, 0x96, 0xc6, 0x13, 0x96, 0x56, \
    0x13, 0x5e, 0x68, 0x75, 0xb9, 0xe9, 0x79, 0x29, \
    0x28, 0x8c, 0x7d, 0xa1, 0xf2, 0x78, 0x7b, 0x66, \
    0x86, 0xcc, 0x9e, 0x6b, 0xf6, 0x03, 0xc2, 0xfe, \
    0x59, 0x1b, 0xab, 0x4a, 0x40, 0x24, 0x70, 0xe4, \
    0x8b
}

```

5.37.3 Building the Demo

- Build Plug & Trust middleware stack. (Refer [Building / Compiling](#))
- SE05X_Auth=None
- Project: se05x_Delete_and_test_provision

5.38 se05x Multiple Digest Crypto Objects example

This project demonstrates managing multiple crypto objects from application layer. The example will create 2 digest crypto objects which will be used for SHA256 multi-step operations. The example can be extended for AES, AEAD, MAC crypto objects also.

Note:

Disable SSSFTR_SE05X_CREATE_DELETE_CRYPTOOBJ in cmake options to run this example. This will ensure crypto objects are not created at SSS layer.

5.38.1 Building the Demo

- Build Plug & Trust middleware stack. (Refer [Building / Compiling](#))
- Project: se05x_MultipleDigestCryptoObj

5.38.2 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_MultipleDigestCryptoObj.exe <PORT NAME>
```

Where **<PORT NAME>** is the VCOM COM port.

On Raspberry-Pi or iMX board, run as:

```
./se05x_MultipleDigestCryptoObj
```

5.38.3 Console output

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

```
sss :WARN :Communication channel is Plain.
sss :WARN :!!!Not recommended for production use.!!!
App :INFO :Running Multiple Digest Crypto Example
    se05x_MultipleDigestCryptoObj.c
App :INFO :Create digest crypto object with id - 58 !!!
App :INFO :Create digest crypto object with id - 59 !!!
App :INFO :Calling Update function for crypto objectId1 !!!
App :INFO :Calling Update function for crypto objectId2 !!!
App :INFO :Calling Update function for crypto objectId1 !!!
App :INFO :Calling Update function for crypto objectId2 !!!
App :INFO :Calling Update function for crypto objectId1 !!!
App :INFO :Calling Update function for crypto objectId2 !!!
App :INFO :Message Digest (input1) successful !!!
App :INFO :digest (Len=32)
    90 B4 6D D6      FA D0 A8 DB      49 69 45 A6      BE 27 D9 5F
    BB 78 60 48      22 35 69 70      56 B8 9B 1D      07 83 68 5E
App :INFO :Message Digest (input2) successful !!!
App :INFO :digest (Len=32)
    D8 ED 22 5A      FC 4B B0 9B      47 EF DB 9E      D4 51 7F 2F
    87 4E 3F 61      00 A3 6D 4F      DD 4D 90 B8      7B 70 72 45
App :INFO :ex_sss Finished
Press any key to continue . . .
```

5.39 SE05X Allow Without SCP example

This project demonstrates how to configure SE05X to allow without platform SCP.

5.39.1 Building the Demo

- Build Plug & Trust middleware stack. (Refer [Building / Compiling](#))
- Project: `se05x_AllowWithoutPlatformSCP`
 - CMake configuration
`SE05X_Auth:PlatfSCP03`

5.39.2 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_AllowWithoutPlatformSCP.exe <PORT NAME>
```

Where <PORT NAME> is the VCOM COM port.

On Raspberry-Pi or iMX board, run as:

```
./se05x_AllowWithoutPlatformSCP
```

5.40 SE05X SCP03 BOOT Example

This project demonstrates that the HostOS does NOT need to know the SCP03 Base Keys to establish an SCP03 session. Provided the boot loader has established the SCP03 session and saved the SCP03 session state.

Refer - simw-top/demos/se05x/se05x_scp03_boot/se05x_scp03_boot.c

5.40.1 Prerequisites

- Build Plug & Trust middleware stack. (Refer [Building / Compiling](#))
- Project: se05x_scp03_boot
- Required build settings

```
cmake -DPTMW_SCP:STRING=SCP03_SSS -DPTMW_SE05X_Auth:STRING=PlatfSCP03 -  
DPTMW_SMCOM:STRING=T1oI2C  
make se05x_scp03_boot
```

5.40.2 Running the Example

Create /tmp/SE05X directory (if not exists) to store the session keys.

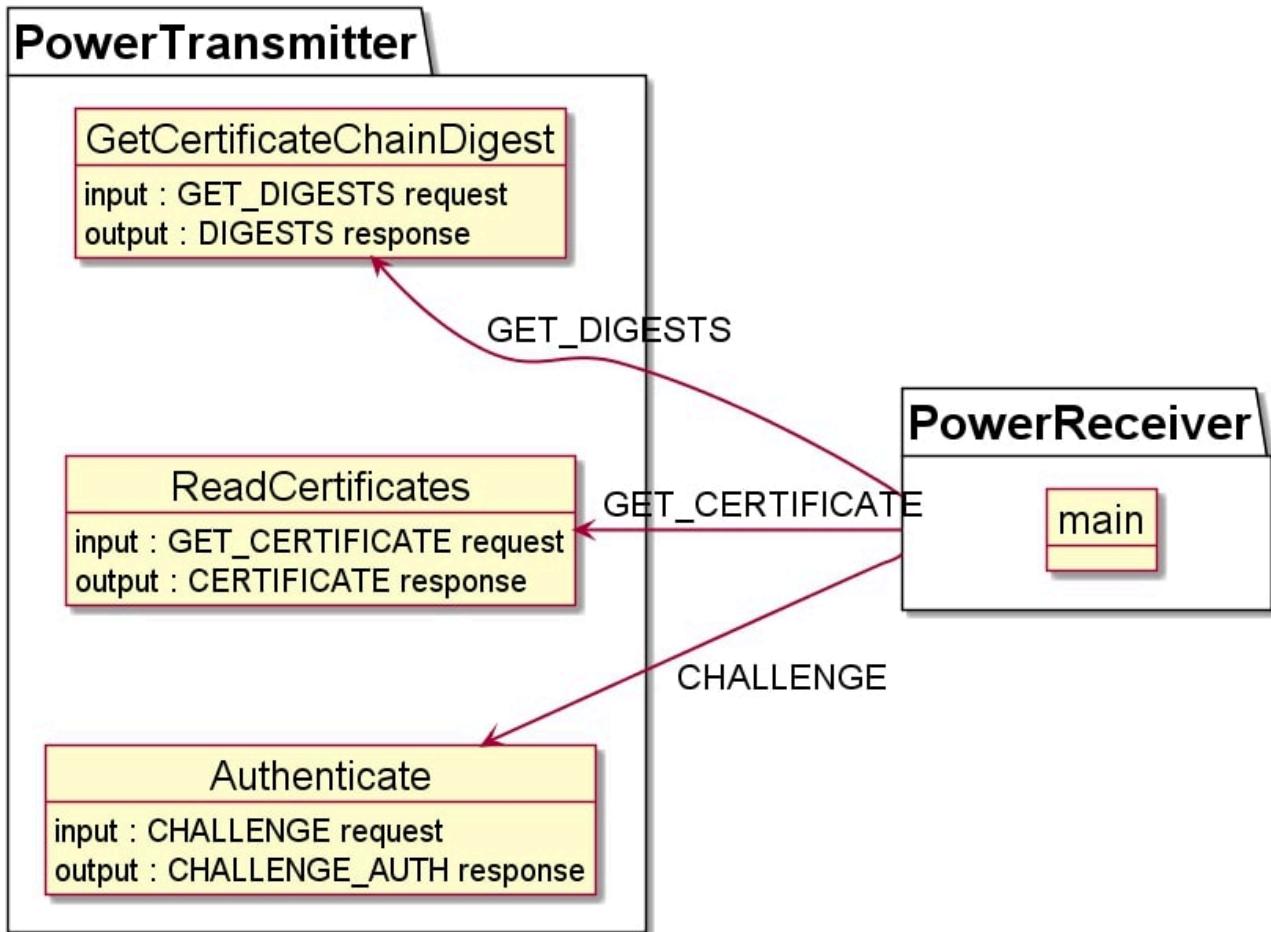
If you have built on Raspberry-Pi or iMX board, run as:

```
./se05x_scp03_boot BOOTLOADER_ROLE  
./se05x_scp03_boot HOST_OS_RESUME
```

5.41 Secure Authenticator (Qi) Authentication demo

- [Pre-requisites](#)
- [GetCertificateChainDigest \(GET_DIGESTS\)](#)
- [ReadCertificates \(GET_CERTIFICATE\)](#)
- [Authenticate \(CHALLENGE\)](#)
- [Building the Demo](#)
- [Running the Example](#)
- [Porting](#)

This project is used to demonstrate the Qi authentication flow between a Power Transmitter and a Power Receiver. The Power Transmitter implements 3 functions for the 3 authentication requests a Receiver can issue to the Transmitter : GetCertificateChainDigest, ReadCertificates, Authenticate.



5.41.1 Pre-requisites

- The secure element should be trust provisioned with correct keys and certificates for Qi Authentication. Keys and certificates can be provisioned for test purpose by updating keys in `demos/se05x/sa_qi_provisioning/sa_qi_credentials.c` and running example [Secure Authenticator \(Qi\) Provisioning demo](#).
- By default WPC Root certificate is used in the certificate chain. If example [Secure Authenticator \(Qi\) Provisioning demo](#) is run, you would need to disable macro `USE_ROOT_WPCCA` in `sa_qi_rootcert.c` to use test RootCA:

```
#ifndef USE_ROOT_WPCCA
#define USE_ROOT_WPCCA 1
#endif
```

5.41.2 GetCertificateChainDigest (GET_DIGESTS)

This function reads the digests of certificate chains stored inside the secure element and returns all the digests as requested by the Power Receiver.

```
void GetCertificateChainDigest(const uint8_t *pGetDigestRequest,
    const size_t getDigestRequestLen,
```

```
uint8_t *pDigestResponse,
size_t *pDigestResponseLen)
{
    smStatus_t retStatus          = SM_NOT_OK;
    uint32_t certChainId         = 0;
    uint16_t objectSize           = 0;
    size_t readSize               = 0;
    uint8_t *pData                = NULL;
    pSe05xSession_t session_ctx = pgSe05xSessionctx;
    uint8_t *pDigestPtr           = NULL;%%CRLF
SP%qi_error_code_t errorCode   = kQiErrorUnspecified;
    uint8_t authMsgHeader        = 0;
    uint8_t requestedSlotMask    = 0;
    uint8_t slotsPopulated       = 0x00;
    uint8_t slotsReturned         = 0x00;
    uint8_t slotsReturnedCount   = 0;%%CRLF
SP%if (NULL == pGetDigestRequest || NULL == pDigestResponse) {
    LOG_E("Null buffer");
    errorCode = kQiErrorInvalidRequest;
    goto error;
}%%CRLF
SP%authMsgHeader = pGetDigestRequest[0];%%CRLF
SP%if (getDigestRequestLen != GET_DIGESTS_CMD_LEN) {
    LOG_E("Invalid request length");
    errorCode = kQiErrorInvalidRequest;
    goto error;
}%%CRLF
SP%requestedSlotMask = pGetDigestRequest[1] & 0x0F;%%CRLF
SP%/* Invalid slot requested */
if (requestedSlotMask == 0) {
    errorCode = kQiErrorInvalidRequest;
    LOG_E("No slot requested");
    goto error;
}%%CRLF
SP%/* Get all populated slots */
retStatus = getPopulatedSlots(session_ctx, &slotsPopulated);
if (SM_OK != retStatus) {
    errorCode = kQiErrorUnspecified;
    LOG_E("Failed to retrieve populated slots");
    goto error;
}%%CRLF
SP%/* Is Slot 0 empty? - Return error as slot 0 must always be populated */
if (!(slotsPopulated & 0x01)) {
    errorCode = kQiErrorUnspecified;
    goto error;
}%%CRLF
SP%/* We will return slots which were requested AND are provisioned */
slotsReturned = requestedSlotMask & slotsPopulated;
pDigestPtr = &pDigestResponse[2];%%CRLF
SP%/* Validate response buffer size */
for (size_t i = 0; i < MAX_SLOTS; i++) {
    if ((slotsReturned) & (0x01 << i)) {
        slotsReturnedCount++;
    }
}
if (*pDigestResponseLen < (size_t)((slotsReturnedCount * DIGEST_SIZE_BYTES)
+ 2)) {
    /* Response buffer size too less */
    errorCode = kQiErrorUnspecified;
    goto error;
}
```

```
 }%%CRLF
SP%%   pData = (uint8_t *)SSS_MALLOC(slotsReturnedCount * DIGEST_SIZE_BYTES);
    if (!pData) {
        errorCode = kQiErrorUnspecified;
        goto error;
    }%%CRLF
SP%%   for (size_t i = 0; i < MAX_SLOTS; i++) {
    if ((slotsReturned) & (0x01 << i)) {
        certChainId = (uint32_t)QI_SLOT_ID_TO_CERT_ID(i);
        /* Read size of object to allocate necessary memory
         * so that we can read the complete object */
        retStatus = Se05x_API_ReadSize(session_ctx, certChainId,
&objectSize);
        if (retStatus != SM_OK) {
            errorCode = kQiErrorUnspecified;
            LOG_E("Failed Se05x_API_ReadSize");
            goto error;
        }
        /* Size of binary object cannot be less than Digest size */
        if (objectSize < DIGEST_SIZE_BYTES) {
            errorCode = kQiErrorUnspecified;
            goto error;
        }
        readSize = DIGEST_SIZE_BYTES;
        retStatus = Se05x_API_ReadObject(session_ctx, certChainId, 0,
DIGEST_SIZE_BYTES, pData, &readSize);
        if (retStatus != SM_OK) {
            errorCode = kQiErrorUnspecified;
            LOG_E("Failed Se05x_API_ReadObject");
            goto error;
        }
        memcpy(pDigestPtr, pData, DIGEST_SIZE_BYTES);
        pDigestPtr += DIGEST_SIZE_BYTES;
    }
}%%CRLF
SP%%   /* Successfully filled all digests - fill response buffer header now */
    pDigestResponse[0] = (uint8_t)(authMsgHeader & 0xF0) + (uint8_t)
(kQiResponseDigest);
    pDigestResponse[1] = (uint8_t)(slotsPopulated << 4) + (uint8_t)
(slotsReturned);
    *pDigestResponseLen = (slotsReturnedCount * DIGEST_SIZE_BYTES) + 2;%%CRLF
SP%%   if (pData) {
        SSS_FREE(pData);
    }%%CRLF
SP%%   return;

error:
    if (pData) {
        SSS_FREE(pData);
    }
    if (pDigestResponse) {
        pDigestResponse[0] = (uint8_t)(authMsgHeader & 0xF0) + (uint8_t)
(kQiResponseError);
        pDigestResponse[1] = (uint8_t)(errorCode);
        pDigestResponse[2] = 0x00;
        *pDigestResponseLen = 3;
    }
}
```

5.41.3 ReadCertificates (GET_CERTIFICATE)

This function reads the certificate chain on the provided slot ID starting from the provided offset and reading provided length bytes.

If the provided offset exceeds 0x600 then that indicates the power transmitter to offset from the Product Unit Certificate. Otherwise the offset starts from the beginning of the certificate chain.

```

void ReadCertificates(const uint8_t *pGetCertificateRequest,
                     const size_t getCertificateRequestLen,
                     uint8_t *pCertificateResponse,
                     size_t *pCertificateResponseLen)
{
    smStatus_t retStatus          = SM_NOT_OK;
    uint16_t objectSize          = 0;
    pSe05xSession_t session_ctx = pgSe05xSessionctx;
    uint32_t certChainId         = 0;
    size_t readSize              = 0;
    uint8_t *pData                = NULL;%%CRLF
SP%%    qi_error_code_t errorCode   = kQiErrorUnspecified;
    uint8_t authMsgHeader        = 0;
    uint8_t requestedslots       = 0;
    uint8_t certificateOffsetA8 = 0;
    uint8_t certificateOffset70 = 0;
    uint8_t certificateLengthA8 = 0;
    uint8_t certificateLength70 = 0;
    uint16_t certificateOffset  = 0;
    uint16_t certificateLength  = 0;
    /* Offset first DIGEST bytes to skip certificate chain hash */
    uint16_t offset               = (uint16_t)DIGEST_SIZE_BYTES;
    uint16_t N_MC                 = 0;
    uint16_t bytesToRead          = 0;%%CRLF
SP%%    if (NULL == pGetCertificateRequest || NULL == pCertificateResponse) {
        LOG_E("Null buffer");
        errorCode = kQiErrorInvalidRequest;
        goto error;
    }%%CRLF
SP%%    authMsgHeader = pGetCertificateRequest[0];%%CRLF
SP%%    if (getCertificateRequestLen != GET_CERTIFICATE_CMD_LEN) {
        LOG_E("Invalid request length");
        errorCode = kQiErrorInvalidRequest;
        goto error;
    }%%CRLF
    requestedslots      = pGetCertificateRequest[1] & 0x03;
    certificateOffsetA8 = (pGetCertificateRequest[1] & 0xE0) >> 5;
    certificateOffset70 = pGetCertificateRequest[2];
    certificateLengthA8 = (pGetCertificateRequest[1] & 0x1C) >> 2;
    certificateLength70 = pGetCertificateRequest[3];
    certificateOffset   = certificateOffsetA8 * 256 + certificateOffset70;
    certificateLength  = certificateLengthA8 * 256 + certificateLength70;%%CRLF
SP%%    certChainId = (uint32_t)QI_SLOT_ID_TO_CERT_ID(requestedslots);%%CRLF
SP%%    retStatus = Se05x_API_ReadSize(session_ctx, certChainId, &objectSize);
    if (retStatus != SM_OK) {
        errorCode = kQiErrorUnspecified;
        LOG_E("Se05x API ReadSize failed");
        goto error;
    }%%CRLF
SP%%    /* Read length of manufacturer certificate to determine the offset for
PUC */
}

```

```
retStatus = getManufacturerCertificateLength(session_ctx, certChainId,
&N_MC);
    if (retStatus != SM_OK) {
        LOG_E("Failed to read manufacturer certificate length");
        errorCode = kQiErrorUnspecified;
        goto error;
} %%CRLF
SP%%    if (certificateOffset >= MAXIMUM_CERT_OFFSET) {
        LOG_D("Read PUC");
        /* N_RH is length of root Hash Certificate and
         * N_MC is length of Manufacturer Certificate
         */
        offset += 2 /* Length of length field */
            /* Length of RootHash */
            + DIGEST_SIZE_BYTES
            /* Length of Manufacturer certificate */
            + N_MC
            /* Offset from Product Unit Certificate */
            + certificateOffset - MAXIMUM_CERT_OFFSET;
}
else {
    offset += certificateOffset;
} %%CRLF
SP%%    /* Calculate actual bytes to read */
if (certificateLength == 0) {
    bytesToRead = objectSize - offset;
}
else {
    bytesToRead = certificateLength;
} %%CRLF
SP%%    if (bytesToRead == 0) {
        /* Cannot read 0 bytes */
        errorCode = kQiErrorInvalidRequest;
        goto error;
} %%CRLF
SP%%    /* Bytes to read cannot exceed the total object size */
if ((offset + bytesToRead) > objectSize) {
    errorCode = kQiErrorInvalidRequest;
    goto error;
} %%CRLF
SP%%    readSize = bytesToRead;
pData = (uint8_t *)SSS_MALLOC(bytesToRead * sizeof(uint8_t));
if (!pData) {
    errorCode = kQiErrorUnspecified;
    goto error;
} %%CRLF
SP%%    if (*pCertificateResponseLen < (size_t)(bytesToRead + 1)) {
        LOG_E("Insufficient buffer");
        errorCode = kQiErrorUnspecified;
        goto error;
} %%CRLF
SP%%    /* Read certificate chain */
retStatus = readCertificateChain(session_ctx, certChainId, offset,
bytesToRead, pData, &readSize);
if (retStatus != SM_OK) {
    errorCode = kQiErrorUnspecified;
    LOG_E("Se05x API ReadObject failed");
    goto error;
}
```

```

LOG_MAU8_D("ReadCertificate object", pData, readSize);%%CRLF
SP%/* Copy the data read out to response buffer */
memcpy(&pCertificateResponse[1], pData, readSize);
if (pData) {
    SSS_FREE(pData);
}%%CRLF
SP% pCertificateResponse[0] = (uint8_t)(authMsgHeader & 0xF0) + (uint8_t)
(kQiResponseCertificate);
*pCertificateResponseLen = (bytesToRead) + 1;%%CRLF
SP% return;

error:
if (pData) {
    SSS_FREE(pData);
}
if (pCertificateResponse) {
    pCertificateResponse[0] = (uint8_t)(authMsgHeader & 0xF0) + (uint8_t)
(kQiResponseError);
    pCertificateResponse[1] = (uint8_t)(errorCode);
    pCertificateResponse[2] = 0x00;
    *pCertificateResponseLen = 3;
}
}
}

```

5.41.4 Authenticate (CHALLENGE)

This function performs the CHALLENGE operation and returns the signature R and signature S values to the power receiver.

```

void Authenticate(const uint8_t *pChallengeRequest,
                  const size_t challengeRequestLen,
                  uint8_t *pChallengeAuthResponse,
                  size_t *pChallengeAuthResponseLen)
{
    smStatus_t retStatus
                = SM_NOT_OK;
    uint16_t objectSize
                = 0;
    pSe05xSession_t session_ctx
                = pgSe05xSessionctx;
    uint8_t *pTbsAuthPtr
                = NULL;
    size_t readSize
                = 0;
    uint8_t certChainHash[DIGEST_SIZE_BYTES]
                = {0};
    uint8_t hash[DIGEST_SIZE_BYTES]
                = {0};
    size_t hashLen
                = sizeof(hash);%%CRLF
SP%    uint8_t authMsgHeader
                = 0;
    uint8_t requestedSlot
                = 0;
    uint8_t slotsPopulated
                = 0x00;
    uint8_t tbsAuth[TBSAUTH_MAX_SIZE]
                = {0};
    size_t tbsAuthlen
                = sizeof(tbsAuth);
    uint8_t signature[MAX_SIGNATURE_LEN]
                = {0};
    size_t sigLen
                = sizeof(signature);
    qi_error_code_t errorCode
                = kQiErrorUnspecified;
    uint32_t certChainId
                = 0;
    uint32_t keyId
                = 0;%%CRLF
SP%    if (NULL == pChallengeRequest || NULL == pChallengeAuthResponse) {
        LOG_E("Null buffer");
        errorCode = kQiErrorInvalidRequest;
        goto error;
}%%CRLF
SP%    authMsgHeader = pChallengeRequest[0];%%CRLF

```

```

SP%%  if (challengeRequestLen != CHALLENGE_CMD_LEN) {
    LOG_E("Invalid request length");
    errorCode = kQiErrorInvalidRequest;
    goto error;
}%%CRLF
SP%%  requestedSlot = pChallengeRequest[1] & 0x03;
certChainId    = (uint32_t)QI_SLOT_ID_TO_CERT_ID(requestedSlot);
keyId         = (uint32_t)QI_SLOT_ID_TO_KEY_ID(requestedSlot);%%CRLF
SP%%  if (*pChallengeAuthResponseLen < CHALLENGE_AUTH_RESPONSE_LEN) {
    LOG_E("Insufficient buffer");
    errorCode = kQiErrorUnspecified;
    goto error;
}%%CRLF
SP%%  retStatus = getPopulatedSlots(session_ctx, &slotsPopulated);
if (SM_OK != retStatus) {
    errorCode = kQiErrorUnspecified;
    LOG_E("Failed to retrieve populated slots");
    goto error;
}%%CRLF
SP%%  if (!(slotsPopulated & 0x01)) {
    /* Slot 0 is empty */
    errorCode = kQiErrorUnspecified;
    goto error;
}
/* Check if the requested slot is populated */
if (!((1 << requestedSlot) & slotsPopulated)) {
    errorCode = kQiErrorInvalidRequest;
    LOG_E("Requested slot not populated");
    goto error;
}%%CRLF
SP%%  retStatus = Se05x_API_ReadSize(session_ctx, certChainId, &objectSize);
if (retStatus != SM_OK) {
    errorCode = kQiErrorUnspecified;
    LOG_E("Se05x_API_ReadSize failed");
    goto error;
}%%CRLF
SP%%  /* Certificate chain size cannot be less than DIGEST_SIZE_BYTES */
if (objectSize < DIGEST_SIZE_BYTES) {
    errorCode = kQiErrorUnspecified;
    goto error;
}%%CRLF
SP%%  /* Read Certificate chain hash value */
pTbsAuthPtr = &tbsAuth[1];
readSize    = DIGEST_SIZE_BYTES;
retStatus   = Se05x_API_ReadObject(session_ctx, certChainId, 0,
DIGEST_SIZE_BYTES, certChainHash, &readSize);
if (retStatus != SM_OK) {
    errorCode = kQiErrorUnspecified;
    LOG_E("Se05x_API_ReadObject failed");
    goto error;
}
memcpy(pTbsAuthPtr, certChainHash, DIGEST_SIZE_BYTES);%%CRLF
SP%%  /* Copy Challenge request to TBS Auth */
pTbsAuthPtr = &tbsAuth[TBSAUTH_CHALLENGE_REQ_OFFSET];
memcpy(pTbsAuthPtr, pChallengeRequest, challengeRequestLen);%%CRLF
SP%%  pChallengeAuthResponse[0] = (uint8_t)(authMsgHeader & 0xF0) + (uint8_t)
(kQiResponseChallengeAuth);
    pChallengeAuthResponse[1] = (uint8_t)(AUTH_PROTOCOL_VERSION << 4) +
(uint8_t)(slotsPopulated);

```

```

    pChallengeAuthResponse[2] = (uint8_t)(certChainHash[DIGEST_SIZE_BYT
%%CRLF
SP%>     tbsAuth[0] = CHALLENGE_AUTH_RESPONSE_PREFIX; // ASCII representation of A
        memcpy(&tbsAuth[TBSAUTH_CHALLENGE_AUTH_RESP_OFFSET],
pChallengeAuthResponse, 3);%%CRLF
SP%> /* Calculate SHA256 of TBSAuth for signature */
retStatus = getSha256Hash(session_ctx, tbsAuth, tbsAuthlen, hash, &hashLen);
if (retStatus != SM_OK) {
    errorCode = kQiErrorUnspecified;
    LOG_E("Failed getSha256Hash");
    goto error;
}%%CRLF
SP%> /* Calculate signature */
retStatus =
    Se05x_API_ECDSSASign(session_ctx, keyId, kSE05x_ECSignatureAlgo_SHA_256,
hash, hashLen, &signature[0], &sigLen);
if (retStatus != SM_OK) {
    LOG_E("sss_asymmetric_sign_digest Failed...");
    errorCode = kQiErrorUnspecified;
    goto error;
}%%CRLF
SP%> /* Extract R and S values from signature */
retStatus = EcSignatureToRandS(signature, &sigLen);
if (retStatus != SM_OK) {
    LOG_E("EcSignatureToRandS Failed...");
    errorCode = kQiErrorUnspecified;
    goto error;
}
*pChallengeAuthResponseLen = (sigLen) + 3;
memcpy(&pChallengeAuthResponse[3], signature, sigLen);%%CRLF
SP%> return;

error:
if (pChallengeAuthResponse) {
    pChallengeAuthResponse[0] = (uint8_t)(authMsgHeader & 0xF0) + (uint8_t)
(kQiResponseError);
    pChallengeAuthResponse[1] = (uint8_t)(errorCode);
    pChallengeAuthResponse[2] = 0x00;
    *pChallengeAuthResponseLen = 3;
}
}
}

```

5.41.5 Building the Demo

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

Select CMake options:

- PTMW_SE05X_Ver=07_02

Note:

If you have run [Secure Authenticator \(Qi\) Provisioning demo](#), do not select PTMW_SE05X_Auth=AESKey

Build project:

- Project: sa_qi_auth

5.41.6 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:

```
sa_qi_auth.exe <PORT NAME>
```

Where **<PORT NAME>** is the VCOM COM port.

On successful execution you should be able to see logs as:

```
App :INFO :PlugAndTrust_v04.01.01_20220112
sss :INFO :atr (Len=35)
    01 A0 00 00 03 96 04 03 E8 00 FE 02 0B 03 E8 00
    01 00 00 00 00 64 13 88 0A 00 65 53 45 30 35 31
    00 00 00

sss :WARN :Communication channel is Plain.
sss :WARN :!!!Not recommended for production use.!!!
App :INFO :Send command GET_DIGESTS
App :INFO :Retrieved digest (Len=32)
    46 29 65 3A D1 CE B3 7C 6A 36 F0 CC 11 B4 29 16
    86 39 27 85 F0 F8 26 DF DE D3 5E AC 5F CC 50 FC

App :INFO :Send command GET_CERTIFICATE
App :INFO :Certificate chain digest successfully verified
App :INFO :Retrieved PUC (Len=442)
    30 82 01 B6 30 82 01 5B A0 03 02 01 02 02 0A 00
    B0 9B 9F 24 00 A7 9E A0 AE 30 0A 06 08 2A 86 48
    CE 3D 04 03 02 30 12 31 10 30 0E 06 03 55 04 03
    0C 07 43 41 43 41 2D 58 31 30 22 18 0F 32 30 32
    31 30 38 32 33 31 35 35 31 35 33 5A 18 0F 32 30
    32 31 30 38 32 34 31 35 35 31 35 33 5A 30 81 8B
    31 2C 30 2A 06 03 55 04 03 0C 23 30 30 30 31 32
    33 2D 52 61 70 69 64 20 63 68 61 72 67 69 6E 67
    20 62 61 67 65 6C 20 74 6F 61 73 74 65 72 31 29
    30 27 06 03 55 04 5C 04 20 53 58 4D 67 64 47 68
    70 63 79 42 68 62 69 42 46 59 58 4E 30 5A 58 49
    67 52 57 64 6E 50 77 3D 3D 31 30 30 2E 06 0A 09
    92 26 89 93 F2 2C 64 01 01 0C 20 44 6F 20 6E 6F
    74 20 75 73 65 20 61 73 20 61 20 66 6C 6F 74 61
    74 69 6F 6E 20 64 65 76 69 63 65 30 59 30 13 06
    07 2A 86 48 CE 3D 02 01 06 08 2A 86 48 CE 3D 03
    01 07 03 42 00 04 07 7B 1F 30 E5 D7 9A 63 FB CC
    35 DE 84 36 E4 5D 89 C1 5F 99 98 E8 B8 F2 C6 00
    1C AE DA E5 F8 59 3A 50 76 D2 C7 A4 AF 0B C5 6B
    47 9D E1 6A DA 11 0C 0A EF D7 39 E1 F0 4D 0D D7
    65 7E B9 32 13 53 A3 1B 30 19 30 17 06 05 67 81
    14 01 02 01 01 FF 04 0B 04 09 F1 02 D3 C4 15 06
    E7 68 79 30 0A 06 08 2A 86 48 CE 3D 04 03 02 03
    49 00 30 46 02 21 00 A8 29 0E C3 C9 AF DC 08 52
    58 CB B4 7A B7 02 3A BC D5 CD 36 78 F4 1D CE 2F
    7C 4C CC 95 46 FC 56 02 21 00 FA EB 67 F9 14 79
    DE 6D 31 DE E7 0B 0C 51 3E 0A 36 1D C4 49 F7 FA
    F4 E9 33 FE 90 49 57 AD EC 10

App :INFO :PUC successfully verified
App :INFO :Manufacturer certificate successfully verified
App :INFO :Certificate chain successfully verified
App :INFO :Retrieved PUC public key (Len=65)
    04 07 7B 1F 30 E5 D7 9A 63 FB CC 35 DE 84 36 E4
    5D 89 C1 5F 99 98 E8 B8 F2 C6 00 1C AE DA E5 F8
```

```

      59 3A 50 76      D2 C7 A4 AF      0B C5 6B 47      9D E1 6A DA
      11 0C 0A EF      D7 39 E1 F0      4D 0D D7 65      7E B9 32 13
      53
App :INFO :Send command CHALLENGE
App :INFO :Challenge Signature (Len=64)
      7E B0 82 D3      3D 91 02 37      AA FE 81 15      DF 02 A2 57
      D1 8C D6 A1      BB C0 20 DE      CF E2 69 B0      57 35 93 5E
      62 E1 E6 37      E0 64 92 5A      D8 BB 7E 92      FB 52 1E 84
      F9 DD A1 43      F9 7B 11 48      0B 1C F9 CD      31 40 77 AD
App :INFO :TBSAuth (Len=54)
      41 46 29 65      3A D1 CE B3      7C 6A 36 F0      CC 11 B4 29
      16 86 39 27      85 F0 F8 26      DF DE D3 5E      AC 5F CC 50
      FC 1B 00 00      00 00 00 00      00 00 00 00      00 00 00 00
      00 00 00 13      11 FC
App :INFO :Challenge successfully verified
App :INFO :ex_sss Finished

```

5.41.7 Porting

The example allows porting of host verification and host RNG functions in case mbedTLS is not available. If you want to add your own implementation of these operations, update the following APIs in `sa_qi_auth/port/sa_qi_helper_port.c`:

```

/* Port to implement RNG to get 16 byte nonce value
 * for authentication operation.
 * This API does not guarantee the randomness of the RNG.
 * User should make sure that the RNG seed is from a trusted source
 * and that the randomness of the source is NIST compliant
 */
int port_getRandomNonce(uint8_t *nonce, size_t *pNonceLen)
{
    int ret          = 0;
    size_t random_length = (*pNonceLen > NONCE_LEN) ? NONCE_LEN : (*pNonceLen);
    *pNonceLen       = random_length;
    ret              = getRandom(nonce, random_length);
    if (0 != ret) {
        *pNonceLen = 0;
    }
    return ret;
}

/* Port to implement function which will
 * parse an X.509 certificate and extract the public key
 * from it.
 */
void port_parseCertGetPublicKey(uint8_t *pCert, size_t certLen, uint8_t
    *pPublicKey, size_t *publicKeylen)
{
    parseCertGetPublicKey(pCert, certLen, pPublicKey, publicKeylen);
}

/* Port to implement function which will
 * verify the complete certificate chain as passed in certificate_chain
 */
int port_hostVerifyCertificateChain(uint8_t *certificate_chain,
    size_t certificate_chain_size,
    uint16_t pucCertOffset,
    uint16_t manufacturerCertLenOffset)

```

```
{  
    return hostVerifyCertificateChain(  
        certificate_chain, certificate_chain_size, pucCertOffset,  
        manufacturerCertLenOffset);  
  
/* Port to implement function which will  
 * verify CHALLENGE on host  
 */  
int port_hostVerifyChallenge(uint8_t *pPublicKey,  
    size_t publicKeyLen,  
    uint8_t *pCertificateChainHash,  
    uint8_t *pChallengeRequest,  
    uint8_t *pChallengeResponse)  
{  
    return hostVerifyChallenge(pPublicKey, publicKeyLen, pCertificateChainHash,  
        pChallengeRequest, pChallengeResponse);  
}
```

5.42 SE052 Reset Example

This project is used to demonstrate the reset of secure element on receiving the APDU throughput error.

The example will send the get random number APDU command in loop to trigger the APDU throughput error. After the error is received, T=1oI2C Chip reset command is invoked using API 'phNxpEse_reset'.

5.42.1 Building the Demo

- Build Plug & Trust middleware stack. (Refer [Building / Compiling](#))
- Project: se052_reset_example

5.42.2 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:

```
se052_reset_example.exe <PORT NAME>
```

Where **<PORT NAME>** is the VCOM COM port.

On Raspberry-Pi or iMX board, run as:

```
./se052_reset_example
```

5.43 SE052 Deep Power down example

This project is used to demonstrate the deep power down on SE052 using T=1oI2C command.

The example will send a proprietary deep power down command (as described in the Datasheet) via I2C link to bring the device into the deep power down mode. The example will wait for 10 seconds before reopening the session which will also make the secure element wakeup from deep power down mode.

Note:

Prerequisites for using deep power down is that the IC core is supplied via Vout like described in AN13904 SE052F - User Guidelines section “Application Circuit I2C Controller (with Deep Power Down)”, as implemented on SE052 demoboard OM-SE052PCB

5.43.1 Building the Demo

- Build Plug & Trust middleware stack. (Refer [Building / Compiling](#))
- Project: se052_deep_pwr_down

5.43.2 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:

```
se052_deep_pwr_down.exe <PORT NAME>
```

Where <PORT NAME> is the VCOM COM port.

On Raspberry-Pi or iMX board, run as:

```
./se052_deep_pwr_down
```

5.44 SE05X Multi Thread and Multi session example

This example is used to demonstrate accessing secure element with multiple threads and each thread with its own session.

5.44.1 Building the Demo

- Build Plug & Trust middleware stack. (Refer [Building / Compiling](#))
- Project: se05x_multi_thread_multi_session

5.44.2 Running the Example

On Raspberry-Pi or iMX board, run as:

```
./se05x_multi_thread_multi_session <PORT_FOR_THREAD1> <PORT_FOR_THREAD2>
```

The example will accept two communication ports used by 2 threads as input arguments. When using the example with access manager, pass the access manager address (default - 127.0.0.1:8040) as both arguments.

Example

```
./se05x_multi_thread_multi_session 127.0.0.1:8040 127.0.0.1:8040
```

5.45 SE05X Access Manager Reset example

This example is used to demonstrate sending the reset command from client applications to access manager module using ‘SM_AmResetI2C’ API.

5.45.1 Building the Demo

- Build Plug & Trust middleware stack. (Refer [Building / Compiling](#))
- Project: se05x_am_reset_ex

5.45.2 Running the Example

On Raspberry-Pi or iMX board, run as:

```
./se05x_am_reset_ex
```

5.46 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 keypairs (ECC and RSA) 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 in case of ECC keys.

5.46.1 Building the example

Use the following CMake configurations to compile the example

- CMake configurations: SSS_HAVE_HOSTCRYPTO_OPENSSL: ON
- Project: seTool

5.46.2 How to use

This example provides four command-line parameters to select the operation to perform.

1. To **generate** an ECC keypair, run the tool as:

```
seTool genECC <keyId> <portname>
```

Where:

- *keyId* is the keypair index at which we want to generate the keypair.

2. To **inject** an ECC keypair, run the tool as:

```
seTool setECC <keyId> <filename> <portname>
```

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 ECC public key**, run the tool as:

```
seTool getECCPublic <keyId> <filename> <portname>
```

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 an ECC reference key** for an injected keypair, run the tool as:

```
seTool getECCRef <keyId> <filename> <portname>
```

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.

5. To **generate** an RSA keypair, run the tool as:

```
seTool genRSA <rsaKeySize> <keyId> <portname>
```

Where:

- *keyId* is the keypair index at which we want to generate the keypair.

6. To **inject** an RSA keypair, run the tool as:

```
seTool setRSA <rsaKeySize> <keyId> <filename> <portname>
```

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.

7. To **retrieve the RSA public key**, run the tool as:

```
seTool getRSAPublic <keyId> <filename> <portname>
```

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.

8. To **create an RSA reference key** for an injected keypair, run the tool as:

```
seTool getRSARef <keyId> <filename> <portname>
```

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.

5.47 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.

5.47.1 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.

5.48 Tests for User Crypto

- Tests are implemented based on pre-computed data.
- Validates cryptographic results against test vectors.
- The User Crypto can be validate with this test suite.

- The tests to be used during User Crypto development.

5.48.1 List of Crypto Algorithms

Following Crypto Algorithms for which tests are available

- Get Random Number
- AES_CBC_Encrypt
- AES_CBC_Decrypt
- MAC_One_Go
- MAC_Multistep

5.48.2 Building the Demo

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

5.49 Ease of Use configuration - Azure IoT Hub

5.49.1 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 ssscli 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/PCWindows/sscli directory.

Refer to section tp-id-range for keyIDs of trust provisioned certificates.

- Read out the trust provisioned certificate as:

```
ssscli connect se05x 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.

5.49.2 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 se05x t1oi2c 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.

5.49.3 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.

5.49.4 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 tp-id-range):

```
#define AZURE_IOT_HUB_NAME "NXPIoTHub"  
#define AZURE_LOCATION_NAME "bengaluru"  
#define AZURE_DEVICE_NAME "528951164068620272177235"
```

```
#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 /7Decimal - 2241349
```

5.49.5 Build and run the demo.

Build and run `cloud_azure`.

CMake configurations:

- RTOS_FreeRTOS: ON
- SSS_HAVE_HOSTCRYPTOMBEDTLS: ON
- SSS_HAVEMBEDTLS_ALT_SSS: ON

5.50 Ease of Use configuration - AWS IoT Console

5.50.1 Pre-requisites

- AWS IoT Console Account
- AWS CLI installed and configured (linked to your AWS account)
- `ssscli` Tool

5.50.2 Extracting Device Certificate

Using `ssscli` Tool, read out the device certificate. Refer to `tp-id-range` 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.

5.50.3 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 file://<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
```

Note:

Additionally, you can create and attach policies using **aws create-policy** and **aws attach-policy** commands. Although this is not required for this demo as we test only publish and subscribe functionalities, you would need to use policies to allow/restrict access to any resource. For more information on policies, refer to [AWS CLI CreatePolicy](#) and [AWS CLI AttachPolicy](#)

5.50.4 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:

```
chmod +x buildScript.sh
./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=<absolute-path-to-MW>/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.

5.50.5 Running on MCU

- 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. Refer to trust provisioned keyIDs listed in `tp-id-range`.

```
#define SSS_KEYPAIR_INDEX_CLIENT_PRIVATE 0x20181005  
#define SSS_CERTIFICATE_INDEX_CLIENT 0x20181007
```

- Build and run project `cloud_aws`.

CMake configurations:

- RTOS_FreeRTOS: ON
- SSS_HAVE_HOSTCRYPTOMBEDTLS: ON
- SSS_HAVEMBEDTLS_ALT_SSS: ON

5.51 SEMS Lite Agent Demo (`sems_lite_ex_update`)

This application will update Applet through SEMS Lite agent.

In this demo, an update package, is pre-compiled into the example binary. See `SEMS_Lite_UpgradeTo_iot_Dev-7_3_0_A8FA.h`. This file contains a byte array that is encoded in protobuf format. This demo will call API `sems_lite_agent_load_package()` which will decode the stream and send update commands command to SE.

Note: This demo is used for OEF A8FA. If you use it for another OEF, you should update the header file with correct one.

5.51.1 Prerequisites

- Micro USB cable
- Kinetis FRDM-K64F/imx-RT1050 board
- Personal Computer
- SE051 Board
- Build Plug & Trust middleware stack. (Refer [Building / Compiling](#))
- Build Project: `sems_lite_ex_update`

5.51.2 Running the Example

- In case the a new update package is available or the IC is a new OEF, replace `SEMS_Lite_UpgradeTo_iot_Dev-7_3_0_A8FA.h`
- Recompile the example for your platform
- If you have built a binary for an embedded target, flash the `sems_lite_ex_update` binary on to the board and reset the board.
- If you have built an exe to be run from PC using VCOM, run as:

```
sems_lite_ex_update.exe <PORT NAME>
```

Where `<PORT NAME>` is the VCOM COM port.

5.52 SEMS Lite CLI APP

This example, is more advanced than [Section 5.12.1 SEMS Lite Agent Demo \(`sems_lite_ex_update`\)](#).

This example takes command line parameters and talks to the SEMS Lite Applet.

Warning:

This application is alpha status and would keep evolving/changing.

5.52.1 Building the CLI APP

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

5.52.2 CLI Parameters

- `--loadpkg <path-to-SEMS-Lite-applet-package-binary-file>`

Load a binary file.

- `--getuid`

Get UID

- `--getappcontents [optional-app-aid]`

Store contents of card to File-Name-to-store-card-contents

e.g. `--getappcontents out_contents_file.bin 01030011`

- `--getpkgcontents [optional-pkg-aid]`

- `--semslitegetversion`

Print version of the SEMS Lite Applet

- `--getsignature <File-Name-to-store-signature>`

Get signature of last script

- `--checkTear`

Check if tearing has happened

- `--checkUpgradeProgress`

Check upgrade progress

- `--getENCIIdentifier`

Get SK.SEMS.ENC Identifier

- `--getCAIdentifier`

Get CA-SEMS Identifier

- `--getCAKeyIdentifier`

Get CA-SEMS signature verification key identifier

- `--getpkgversion`

Get Package Version

- `--getFreePHeap`

Get PHeap Information

- `--getECParameter`

Get configured EC domain parameter type

- --getFIPSInfo

Get configured FIPS Information

- --testapplet <applet-aid> <apdu-command>

Used to select test applet and send dummy APDU Command

5.52.3 Usage - K64F

- Ensure VCOM interface is setup for K64.
- SET EX_SSS_BOOT_SSS_PORT=<COM Port>
- Run application with applicable commandline parameters

5.52.4 Usage - Raspberry Linux

- Run application with applicable commandline parameters

5.52.5 Version Compatibility

SEMS Lite cli application from SEMS Lite 2.0.0 can support binary file generated by tools of SEMS Lite 2.0.0 and those of SEMS Lite 1.0.0. But SEMS Lite cli application from SEMS Lite 1.0.0 can not support binary file generated by tools of SEMS Lite 2.0.0.

	Bin file from SEMS Lite 1.0.0	Bin file from SEMS Lite 2.0.0
Cli app from SEMS Lite 1.0.0	Supported	Not supported
Cli app from SEMS Lite 2.0.0	Supported	Supported

5.53 Key Injection to PUF

This example demonstrates how to enroll PUF on LPC55S, inject PlatformSCP keys into PUF and retrieve key codes. This example can be used as a starting point to inject default SCP03 keys into PUF.

Note:

After running this example, update ex_scp03_puf.h file with the new Activation code and keyCodes.

5.53.1 Pre-requisites

- [Section 10.6.5LPC55S69](#)
- [Section 4.2.4Logging on console](#)
- Build Plug & Trust middleware stack. (Refer [Building / Compiling](#))

5.53.2 How to build

- Replace the following keys with the keys to be provisioned into PUF:

```
/** New key material
 * These will be the static platform SCP03 keys
 * which will be provisioned on the SE and in PUF.
 */
uint8_t PROV_KEY_ENC[PUF_INTRINSIC_KEY_SIZE] = SSS_AUTH_KEY_ENC;
uint8_t PROV_KEY_MAC[PUF_INTRINSIC_KEY_SIZE] = SSS_AUTH_KEY_MAC;
uint8_t PROV_KEY_DEK[PUF_INTRINSIC_KEY_SIZE] = SSS_AUTH_KEY_DEK;
```

For information on PUF, refer [SCP03 with PUF](#).

- Compile and run the example with the following CMake options:
 - Host=lpcxpresso55s
 - Project:puf_inject_scp03

5.53.3 How to use

- Flash the binary on the device.
- On successful execution, you will be able to see the ActivationCode and KeyCodes printed out on the console.

5.53.4 Injecting keys into PUF

Refer to the below implementation on how to implement a simple function to inject SCP03 keys into PUF.

```
static status_t puf_insert_scp03_keys(uint8_t *PROV_KEY_ENC, uint8_t
    *PROV_KEY_MAC, uint8_t *PROV_KEY_DEK)
{
    status_t result = kStatus_Fail;
    uint8_t activationCode[PUF_ACTIVATION_CODE_SIZE];%%CRLF
SP%%    srand(0xbababeda);%%CRLF
SP%%    puf_config_t conf;
    PUF_GetDefaultConfig(&conf);
    PUF_Deinit(PUF, &conf);%%CRLF
SP%%    result = PUF_Init(PUF, &conf);
    ENSURE_OR_GO_CLEANUP(result == kStatus_Success);%%CRLF
SP%%    result = PUF_Enroll(PUF, activationCode, sizeof(activationCode));
    ENSURE_OR_GO_CLEANUP(result == kStatus_Success);
    LOG_MAU8_I("ActivationCode", activationCode, sizeof(activationCode));
    PUF_Deinit(PUF, &conf);%%CRLF
SP%%    result = PUF_Init(PUF, &conf);
    ENSURE_OR_GO_CLEANUP(result == kStatus_Success);%%CRLF
SP%%    result = PUF_Start(PUF, activationCode, PUF_ACTIVATION_CODE_SIZE);
    ENSURE_OR_GO_CLEANUP(result == kStatus_Success);%%CRLF
SP%%    result = PUF_SetUserKey(PUF,
        kPUF_KeyIndex_00,
        PROV_KEY_ENC,
        PUF_INTRINSIC_KEY_SIZE,
        keyCodeENC_01,
        PUF_GET_KEY_CODE_SIZE_FOR_KEY_SIZE(PUF_INTRINSIC_KEY_SIZE));
    ENSURE_OR_GO_CLEANUP(result == kStatus_Success);
    LOG_MAU8_I("KeyCode_ENC", keyCodeENC_01,
    PUF_GET_KEY_CODE_SIZE_FOR_KEY_SIZE(PUF_INTRINSIC_KEY_SIZE));
    result = PUF_SetUserKey(PUF,
        kPUF_KeyIndex_00,
        PROV_KEY_MAC,
        PUF_INTRINSIC_KEY_SIZE,
        keyCodeMAC_01,
        PUF_GET_KEY_CODE_SIZE_FOR_KEY_SIZE(PUF_INTRINSIC_KEY_SIZE));
    ENSURE_OR_GO_CLEANUP(result == kStatus_Success);
    LOG_MAU8_I("KeyCode_MAC", keyCodeMAC_01,
    PUF_GET_KEY_CODE_SIZE_FOR_KEY_SIZE(PUF_INTRINSIC_KEY_SIZE));
    result = PUF_SetUserKey(PUF,
        kPUF_KeyIndex_00,
        PROV_KEY_DEK,
        PUF_INTRINSIC_KEY_SIZE,
        keyCodeDEK_01,
```

```
PUF_GET_KEY_CODE_SIZE_FOR_KEY_SIZE( PUF_INTRINSIC_KEY_SIZE ) ;  
ENSURE_OR_GO_CLEANUP( result == kStatus_Success );  
LOG_MAU8_I( "KeyCode_DEK", keyCodeDEK_01,  
PUF_GET_KEY_CODE_SIZE_FOR_KEY_SIZE( PUF_INTRINSIC_KEY_SIZE ) );  
  
cleanup:  
    PUF_Deinit( PUF, &conf );  
    return result;  
}
```

5.54 Key Rotation using PUF

This example demonstrates how to use PUF to manage PlatformSCP keys and rotate the keys using PUF. For details on PUF and usage with LPC55S, refer to [SCP03 with PUF](#).

Before running this example, be sure that correct PlatformSCP keys are already provisioned in PUF. For details on how to provision keys in PUF, refer [Section 5.13.1Key Injection to PUF](#).

In this example, we first open a session with default PlatformSCP keys and perform an RNG operation, then we rotate the keys in SE and PUF, reopen session with new keys and perform RNG operation again to demonstrate that the keys have been rotated. Finally, we revert to the old keys.

Warning:

We are using randomized keys for key rotation. Make sure that the demo runs completely without any power interruptions. In case of failure, SE050 could be using the new keys and re-running the demo will fail.

5.54.1 Pre-requisites

- [Section 10.6.5LPC55S69](#)
- [Section 4.2.4Logging on console](#)
- Build Plug & Trust middleware stack. (Refer [Building / Compiling](#))
- PUF must be enrolled first and original SCP03 keys must be provisioned and ActivationCode and KeyCodes must be updated in `ex_scp03_puf.h`. SBL and secure app should be compiled with the correct AC and KCs. See [Section 5.13.1Key Injection to PUF](#) on how to provision PUF with PlatformSCP03 keys.

5.54.2 How to build

Make sure that you compile the secure example first. The non-secure example links to the secure example.

Compile the secure example with the following CMake options:

- Host=lpcxpresso55s_s
- SCP=SCP03_SSS
- SE05X_Auth=PlatfSCP03
- Project:puf_rotate_scp03_s

Compile the non-secure example with the following CMake options:

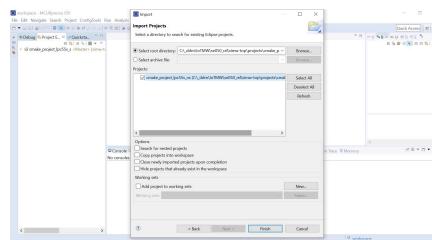
- Host=lpcxpresso55s_ns
- SCP=SCP03_SSS
- SE05X_Auth=PlatfSCP03
- Project:puf_rotate_scp03_ns

5.54.3 How to run

Follow the steps given below to flash secure and non-secure binaries on LPC55S board.

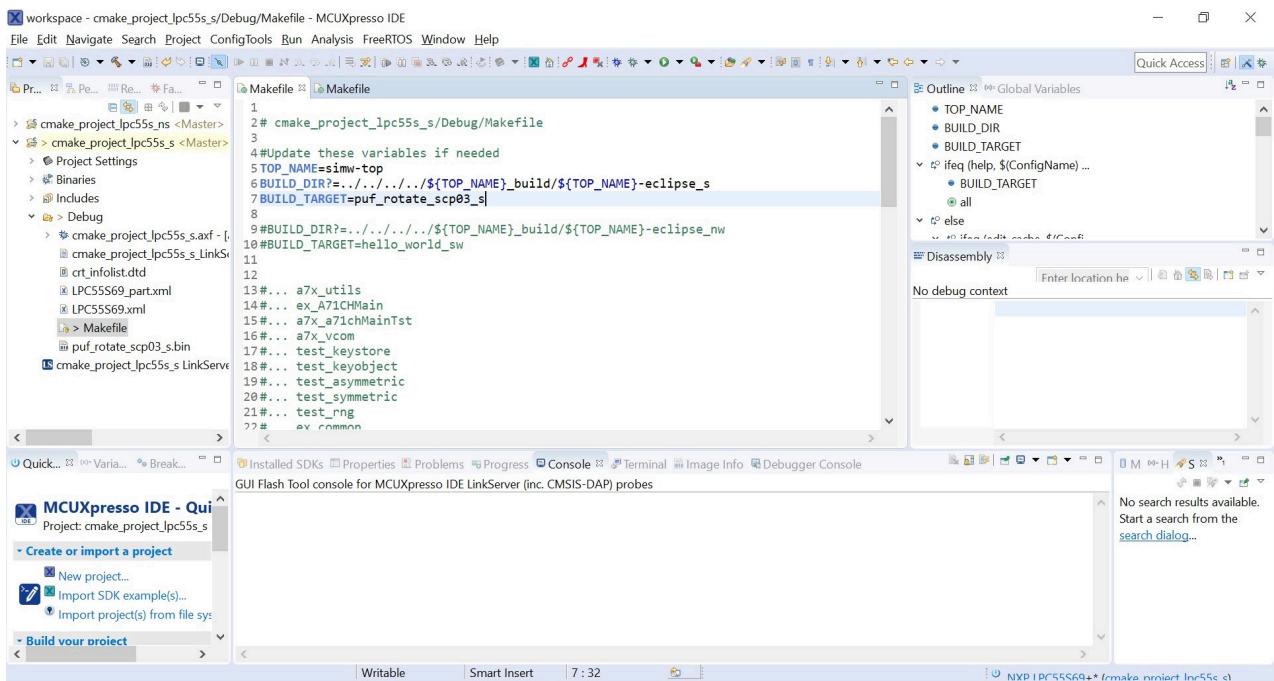
1.

Import secure and non-secure projects into MCUXpresso IDE



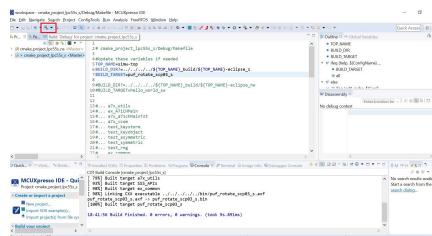
2.

Update Makefile target for both projects



3.

Build the projects.



Note:

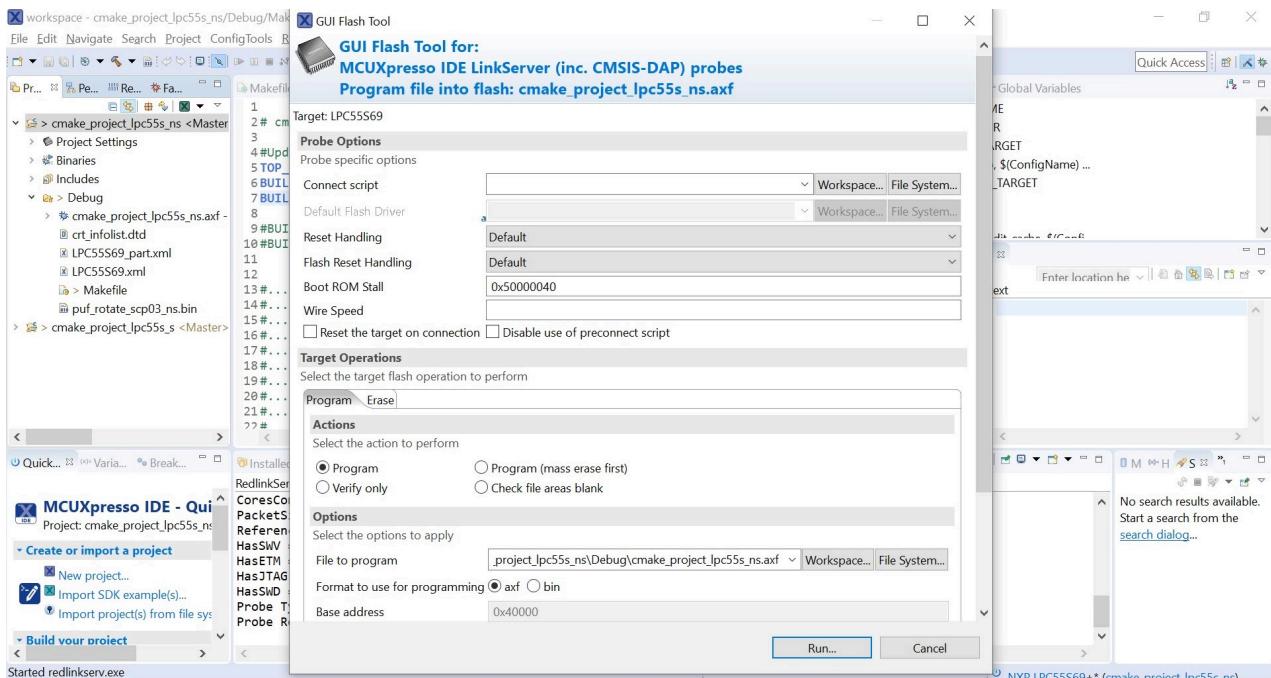
Be sure that you build the secure project first and then the non-secure project.

4.

Start GUI Flash Tool

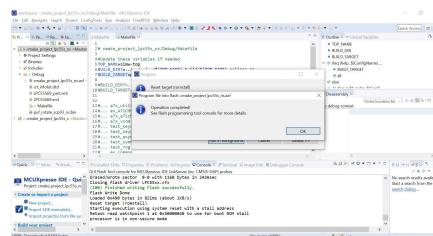
Note:

You can program the binary by debugging the project also. If you want to debug, go to step 6.



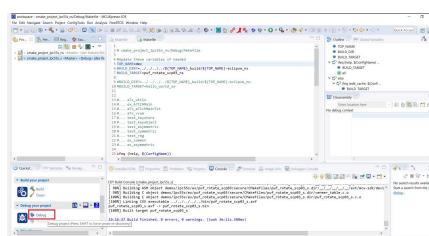
5.

On successful operation you should see the following message



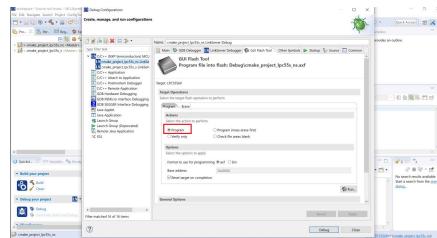
6.

To start debugging into the project, simply select the project that you want to debug and press the Debug button in QuickStart Menu.



7.

Make sure that in the Debug Configuration under GUI Flash Tool tab, you have selected **Program**.



Perform the last two steps for both the projects (order does not matter). While debugging, flash the program that you want to debug second.

When you have flashed both the projects, reset the board. On successful execution you would be able to see the following log in terminal

```

App :INFO :PlugAndTrust_v02.15.00_20200522
sss :INFO :atr (Len=35)
      01 A0 00 00      03 96 04 03      E8 00 FE 02      0B 03 E8 00
      01 00 00 00      00 64 13 88      0A 00 65 53      45 30 35 31
      00 00 00
sss :INFO :atr (Len=35)
      01 A0 00 00      03 96 04 03      E8 00 FE 02      0B 03 E8 00
      01 00 00 00      00 64 13 88      0A 00 65 53      45 30 35 31
      00 00 00
App :INFO :Applet selection successful!
App :INFO :Random test 1 was successful, with default SCP03 keys!
sss :INFO :atr (Len=35)
      01 A0 00 00      03 96 04 03      E8 00 FE 02      0B 03 E8 00
      01 00 00 00      00 64 13 88      0A 00 65 53      45 30 35 31
      00 00 00
App :INFO :Applet deselection successful!
App :INFO :Key Rotation was successful!
sss :INFO :atr (Len=35)
      01 A0 00 00      03 96 04 03      E8 00 FE 02      0B 03 E8 00
      01 00 00 00      00 64 13 88      0A 00 65 53      45 30 35 31
      00 00 00
App :INFO :Applet selection successful!
App :INFO :Applet is now using PUF keys!
App :INFO :Random test 2 was successful, with new PUF keys!
sss :INFO :atr (Len=35)
      01 A0 00 00      03 96 04 03      E8 00 FE 02      0B 03 E8 00
      01 00 00 00      00 64 13 88      0A 00 65 53      45 30 35 31
      00 00 00
App :INFO :Applet deselection successful!
App :INFO :Key Rotation was successful!
App :INFO :Rotation back to default keys was successful!
App :INFO :Entering normal world.

Welcome in normal world (SIMW) !

```

5.55 Secure Boot Demo

This example is to demonstrate how secure binding is achieved between LPC55S and SE05X using PUF for secure storage of PlatformSCP keys on the MCU. For details on secure boot process, refer to [Section 3.19Secure Boot](#)

5.55.1 Secondary Bootloader (SBL)

Refer to [Section 3.19.2Secondary Bootloader](#) for details on operations performed by the Secondary Bootloader (SBL).

5.55.2 Secure Application

The secure application is verified by SBL using RSA2k public key stored at KeyID K_PUB_OEM_ID. Secure application will have the reference to keyCodes prepared by SBL, using which PlatformSCP session can be established.

In this example, secure application is opening a PlatformSCP03 session using keyCodes prepared by SBL and transfers the control to non-secure application which will open an Applet session and test communication.

5.55.3 Non-Secure Application

The non-secure application used in this example is used to establish an Applet session and use non-secure callable APIs provided in secure application to wrap data with PlatformSCP context before sending to the SE.

5.55.4 Prerequisites

- [Section 10.6.5LPC55S69](#)
- [Section 4.2.4Logging on console](#)
- Building Plug & Trust middleware stack. (Refer [Building / Compiling](#))
- PUF must be enrolled first and original SCP03 keys must be provisioned and ActivationCode and KeyCodes must be updated in ex_scp03_puf.h. SBL and secure app should be compiled with the correct AC and KCs. See [Section 5.13.1Key Injection to PUF](#) on how to provision PUF with PlatformSCP03 keys.
- Secure application verification public key must be provisioned at keyID K_PUB_OEM_ID and updated in puf_se05x_sbl_s.c at:

```
#define K_PUB_OEM_ID 0xA55A
```

Refer to [CLI Tool](#) for details on how to provision keys into SE05x.

- New PlatformSCP03 keys updated in puf_se05x_sbl_s.c at:

```
uint8_t newScpKeyENC[PUF_INTRINSIC_KEY_SIZE] = {0};  
uint8_t newScpKeyMAC[PUF_INTRINSIC_KEY_SIZE] = {0};  
uint8_t newScpKeyDEK[PUF_INTRINSIC_KEY_SIZE] = {0};  
  
status = getHostAesKeys(pCtx, newScpKeyENC, PUF_INTRINSIC_KEY_SIZE);  
ENSURE_OR_GO_CLEANUP(status == kStatus_SSS_Success);  
// LOG_MAU8_W("Random ENC Key", newScpKeyENC, PUF_INTRINSIC_KEY_SIZE);  
  
status = getHostAesKeys(pCtx, newScpKeyMAC, PUF_INTRINSIC_KEY_SIZE);  
ENSURE_OR_GO_CLEANUP(status == kStatus_SSS_Success);  
// LOG_MAU8_W("Random MAC Key", newScpKeyMAC, PUF_INTRINSIC_KEY_SIZE);  
  
status = getHostAesKeys(pCtx, newScpKeyDEK, PUF_INTRINSIC_KEY_SIZE);  
ENSURE_OR_GO_CLEANUP(status == kStatus_SSS_Success);  
// LOG_MAU8_W("Random DEK Key", newScpKeyDEK, PUF_INTRINSIC_KEY_SIZE);
```

Note:

The SBL performs key rotation when run first time with randomized keys. If required for development, the user should change this to known keys so that in case of failure recovering SE050 is easy.

- elftosb tool from [ELF to Secure Binary GUI](#) to prepare signed images.
- Also see [Section 3.19 Secure Boot](#)
- Refer AppNote [AN12283](#).

5.55.5 How to build

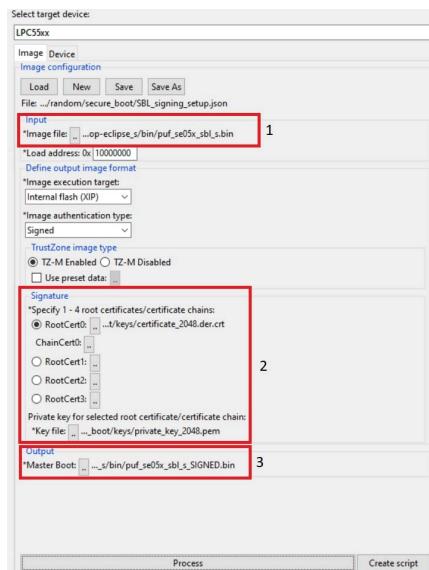
1. Compile the SBL with the following CMake options:

- PTMW_Host=lpcxpresso55s_s
- PTMW_SCP=SCP03_SSS
- PTMW_SE05X_Auth=PltfSCP03
- PTMW_SBL:STRING=None
- Project:puf_se05x_sbl_s

2.

To complete the build process, you also need to sign the image for Bootloader verification. For this, you need to use [ELF to Secure Binary GUI](#).

Configure elftoss GUI tool as:



- Select your SBL input binary
- Add certificate chain and signing key (Refer [Chapter 3. Keys and certificates](#) in [AN12283](#))
- Select your output signed SBL binary

Press **Process** button to create signed image.

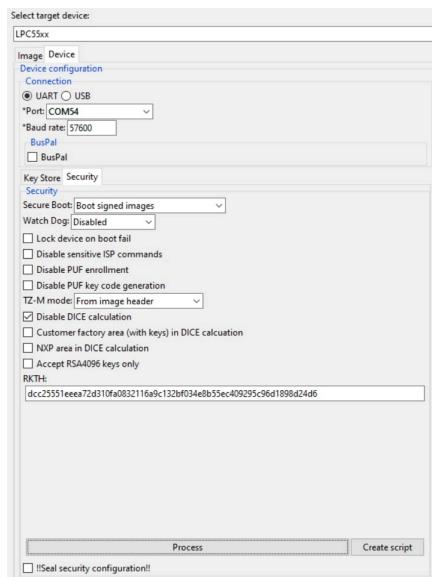
You should be able to see the log generated on right window containing RKTH as highlighted:

Note this value for next step.

3.

We will configure CMPA and CFPA pages in next 2 steps. These pages are required to configure LPC55S registers before boot. We can configure boot config, boot speed, debugging registers, RKTH, etc (See [AN12283](#) and [AN13037](#) for more details). In this example, we configure CMPA page with RKTH value and CFPA with RoT Keys enabled. RKTH value is checked from CMPA while verifying the image to ensure that a different certificate chain was not used to create the signed binary. CFPA page is used to check if any key used in signed image preparation is enabled or revoked.

In Device tab, configure the following settings:



In place of RKTH, enter your RKTH that you noted in the previous step.

Note:

Do NOT Seal Security configuration

Enter ISP mode by holding ISP button on the device and pressing Reset button.

Update the COMPort and press Process. This will update CMPA page with this RKTH value which will allow ROM Bootloader to successfully verify SBL image. On successful programming of CMPA page, you should be able to see logs like this:

```
[2021-05-11 08:25] >|jshost|win10\host.exe -v PCOM54,57600 -read-memory 0x8E400 512 |itemspcConfOrGib  
[2021-05-11 08:25] >|jshost|win10\host.exe -v PCOM54,57600 -options =0x0  
Inject command "read-memory"  
Successful response to command "read-memory"  
Successful generic response to command "read-memory"  
Response status = 0 (OK) Success.  
Read 512 of 512 bytes.  
[2021-05-11 08:26] >|jshost|win10\host.exe -v PCOM54,57600 -write-memory 0x8E400 512 |itemspcConfMod.bin  
[2021-05-11 08:26] >|jshost|win10\host.exe -v PCOM54,57600 -options =0x0  
Inject command "write-memory"  
Successful response to the target bytes to the target.  
Successful response to command "get-property(max-packet-size)"  
(1/1)1094 Completed.  
Successful generic response to command "write-memory"  
- Took 0.115 seconds  
Response status = 0 (OK) Success.  
Write 512 of 512 bytes.
```

Note:

This will allow to boot only signed images. You won't be able to boot plain images. To revert this setting to boot plain images, change Secure Boot setting to **Boot plain images** and update the CMPA page again.

4.

Update CFPA page using `blhost` utility available in `elftosb` package:

```
blhost.exe -p COM54 write-memory 0x9DE00 C:\_ddm\PnP\random\secure_boot  
    \AN12283\CFPA_0x9de00.bin
```

```
C:\Windows\system32\cmd.exe /p C:\Windows\Temp\random\secure_boot\AN22283\CP0x_0x0d00.bin> !random!random!boot!boot.exe -p C:\Windows\Temp\random\secure_boot\AN22283\CP0x_0x0d00.bin> write memory 0x001000 C:\Windows\Temp\random\secure_boot\AN22283\CP0x_0x0d00.bin> write memory 0x001000 C:\Windows\Temp\random\secure_boot\AN22283\CP0x_0x0d00.bin> inject command "write-memory"> preparing to send 0x001000 bytes to the target.> sending generic response to command "write-memory".> [1] write-memory completed> successful generic response to command "write-memory".> response status = 0 (0x0) Success.> wrote 512 of 512 bytes
```

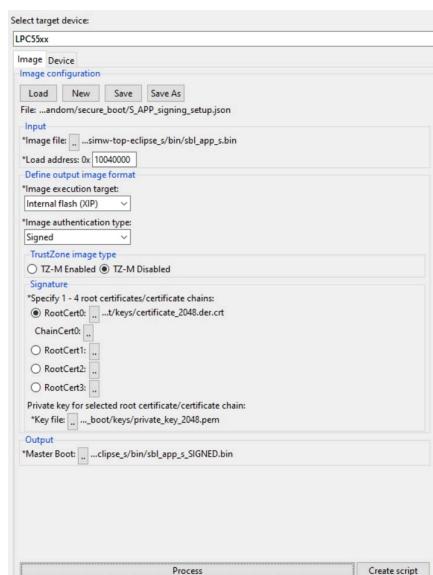
Sample CFPA page is available with this demo which can be directly flashed. If you want to update the CFPA page again, you need to increment the CFPA version. See [AN12283](#) for more details.

5. Compile SBL bootable secure application with the following CMake options:

- PTMW_Host=lpcxpresso55s_s
 - PTMW SCP=SCP03_SSS
 - PTMW_SE05X_Auth=PlatfSCP03
 - PTMW_SBL=SBL_LPC55S
 - Project:sbl app s

6.

To prepare Secure application signed image, configure elftosb as:



Run process to create signed secure application image.

7. Compile non-secure application with the following CMake options:

- PTMW_Host=lpcxpresso55s_ns
- PTMW SCP=SCP03_SSS
- PTMW_SE05X_Auth=PlatfSCP03
- PTMW_Log=Silent
- Project:sbl_app_ns

Note:

This process creates a signed image for SBL. However, there is an option to create signed + encrypted image. In that, the encrypted image is first decrypted by symmetric key provisioned in PUF, then the decrypted image is loaded in flash for signature verification. Refer to section 5.6 in [AN12283](#) for details on creating Secure Binary (Signed + Encrypted).

A "Secure" Binary in this case means it is signed and encrypted, it is not related to Secure Boot or LPC55S TrustZone.

Using a Secure Binary would also disable debugging. Refer to [AN13037](#) for details on how to update CC_SOCU registers in CMPA and CFPA pages to enable debugging.

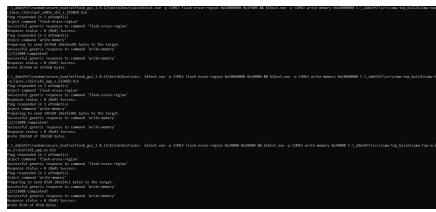
5.55.6 How to run

Enter ISP mode by holding ISP button on the device and pressing Reset button. Flash the built images with these commands:

```
blhost.exe -p COMxx flash-erase-region 0x10000000 0x3F000 && blhost.exe -p COMxx
write-memory 0x10000000 puf_se05x_sbl_s_SIGNED.bin

blhost.exe -p COMxx flash-erase-region 0x10040000 0x30000 && blhost.exe -p COMxx
write-memory 0x10040000 sbl_app_s_SIGNED.bin

blhost.exe -p COMxx flash-erase-region 0x70000 0x28000 && blhost.exe -p COMxx
write-memory 0x70000 sbl_app_ns.bin
```



Press the Reset button after completion. On successful execution you should be able to see these logs:

First boot log:

```
App :INFO :PlugAndTrust_v03.01.00_20210102
sss :INFO :atr (Len=35)
      01 A0 00 00      03 96 04 03      E8 00 FE 02      0B 03 E8 00
      01 00 00 00      00 64 13 88      0A 00 65 53      45 30 35 31
      00 00 00

App :INFO :Applet selection successful!
App :INFO :Random test was successful with default SCP03 keys
sss :INFO :atr (Len=35)
```

```

          01 A0 00 00      03 96 04 03      E8 00 FE 02      0B 03 E8 00
          01 00 00 00      00 64 13 88      0A 00 65 53      45 30 35 31
          00 00 00

App :INFO :Applet deselection successful!
App :INFO :Key Rotation was successful!
App :INFO :NewKeyCode_ENC (Len=52)
          00 00 00 02      E2 26 23 39      CB 6F D3 57      46 75 7D 7E
          47 9C F4 68      D6 65 D4 8C      3D CF 28 4F      1F 3F 9D 4A
          66 1F E3 D1      BB B0 0A D7      57 C3 35 3B      48 17 1B AF
          33 AE 2C A6

App :INFO :NewKeyCode_MAC (Len=52)
          00 00 00 02      10 F9 83 8B      91 BD 14 5E      6D 57 4A C2
          46 13 7A 71      38 8D 14 05      99 B3 08 13      EE 1B 68 9E
          36 E6 99 4B      EC 1B BB 48      2D 50 58 D1      16 6E F3 3D
          8D 2F 18 41

App :INFO :NewKeyCode_DEK (Len=52)
          00 00 00 02      4D F1 02 83      E7 B9 C5 30      66 DB 92 43
          86 7E 3C E2      A3 61 FF 48      72 C9 AE 68      03 31 6C 7B
          6F C9 5C E4      28 BC D1 3F      B6 DD 6C C9      AF 65 D8 B0
          0B 38 E2 E6

sss :INFO :atr (Len=35)
          01 A0 00 00      03 96 04 03      E8 00 FE 02      0B 03 E8 00
          01 00 00 00      00 64 13 88      0A 00 65 53      45 30 35 31
          00 00 00

App :INFO :Applet selection successful!
App :INFO :Session Open successful
App :INFO :Random test was successful with new SCP03 keys
App :INFO :Image verification successful, booting the application now!
sss :INFO :atr (Len=35)
          01 A0 00 00      03 96 04 03      E8 00 FE 02      0B 03 E8 00
          01 00 00 00      00 64 13 88      0A 00 65 53      45 30 35 31
          00 00 00

App :INFO :Applet deselection successful!
sss :INFO :atr (Len=35)
          01 A0 00 00      03 96 04 03      E8 00 FE 02      0B 03 E8 00
          01 00 00 00      00 64 13 88      0A 00 65 53      45 30 35 31
          00 00 00

App :INFO :Applet selection successful!
App :INFO :Random test was successful from secure application
App :INFO :Entering normal world see you there.

Non-secure entry
NS Channel initialize successful
sss:WARN :Communication channel is Plain.
sss:WARN :!!!Not recommended for production use.!!!
NS Session open is Successful
sss_rng_get_random successful

```

Next boot log:

```

App :INFO :PlugAndTrust_v03.01.00_20210102
sss :INFO :atr (Len=35)
          01 A0 00 00      03 96 04 03      E8 00 FE 02      0B 03 E8 00
          01 00 00 00      00 64 13 88      0A 00 65 53      45 30 35 31
          00 00 00

App :INFO :Applet selection successful!
App :INFO :Random test was successful, with KCs loaded from Flash!
App :INFO :Image verification successful, booting the application now!
sss :INFO :atr (Len=35)
          01 A0 00 00      03 96 04 03      E8 00 FE 02      0B 03 E8 00

```

```
          01 00 00 00      00 64 13 88      0A 00 65 53      45 30 35 31
          00 00 00
App   :INFO :Applet deselection successful!
sss   :INFO :atr (Len=35)
          01 A0 00 00      03 96 04 03      E8 00 FE 02      0B 03 E8 00
          01 00 00 00      00 64 13 88      0A 00 65 53      45 30 35 31
          00 00 00
App   :INFO :Applet selection successful!
App   :INFO :Random test was successful from secure application
App   :INFO :Entering normal world see you there.

Non-secure entry
NS Channel initialize successful
sss:WARN :Communication channel is Plain.
sss:WARN :!!!Not recommended for production use.!!!
NS Session open is Successful
sss_rng_get_random successful
```

6 NXP EdgeLock 2GO Agent

6.1 Introduction

EdgeLock 2GO is a cloud service by NXP for provisioning keys and credentials into devices equipped with SE050 and for easily onboarding the device into the cloud services of the user. Please visit <https://www.nxp.com/edgelock2go> for more information.

The EdgeLock 2GO agent is the on-device counterpart of the EdgeLock 2GO cloud service. Its purpose is to establish a secure connection to the EdgeLock 2GO service, report status of the device and update the device with up-to-date credentials and configuration data. It handles credentials for authentication at customer cloud services in cooperation with a secure keystore and manages configuration data/connection information for these the cloud services.

6.2 Building and running the EdgeLock 2GO agent

6.2.1 Building / Compiling the EdgeLock 2GO agent

The build instructions for the EdgeLock 2GO agent do not deviate from the build instructions already introduced in section [Building / Compiling](#). For convenience, the script `<SE05X_root_folder>/simw-top/scripts/create_cmake_projects.py` will generate bespoke CMake configurations with all CMake options set correctly for building the EdgeLock2 GO agent for MCU-SDK (FRDMK64F, LPC55S69), and i.MX:

- KSDK: `<SE05X_root_folder>/simw-top_build/simw-top-eclipse_arm_el2go`
- i.MX (native compilation): `<SE05X_root_folder>/simw-top_build/imx_native_se050_t1oi2c_openssl_el2go`

In case of Unix builds the el2go_agent uses PAHO library to execute MQTT connection tests. In this case is important that the user correctly sets the values of the PAHO cmake cache variables PAHO_BUILD_SHARED and PAHO_BUILD_STATIC. In case the el2go_agent is built with usage of shared libraries (-DWithSharedLIB=ON), then the following options should be chosen when running cmake command: -DPAHO_BUILD_SHARED=TRUE -DPAHO_BUILD_STATIC=FALSE. In case of build with static libraries use the following options instead: -DPAHO_BUILD_SHARED=FALSE -DPAHO_BUILD_STATIC=TRUE.

6.2.2 Registering the device to the EdgeLock 2GO service

In order to connect your device to EdgeLock 2GO and provision the keys and credentials that you have configured, you first need to register your device to your EdgeLock 2GO account. This can be done in different ways including:

- Registering your device UID into your EdgeLock 2GO account. You must first read-out the UID of your device. This can be achieved for example by executing the se05x_Get_Info executable that is part of this release. How to do this is described in more detail in [SE05X Get Info example](#).
- Injecting a claim code on the device, see [Claim Codes](#).

For more details, please refer to the EdgeLock 2GO documentation (AN12691).

6.2.3 Connecting the device to the EdgeLock 2GO service

For the connection to the EdgeLock 2GO cloud service, each account uses an individual hostname. You can obtain this hostname in the company settings on the GUI of the EdgeLock 2GO cloud service. The hostname can be picked up by the EdgeLock 2GO agent from different locations, please see also [Parameters for the connection to EdgeLock 2GO cloud service](#).

Once you have registered your device or installed a claim code, you can simply connect your device to the EdgeLock 2GO cloud service by calling the EdgeLock 2GO agent API. See [EdgeLock 2GO Agent Examples](#) for an example. During the connection to the EdgeLock 2GO cloud service, the device will get provisioned with the credentials that you have configured. For more details, please refer to the EdgeLock 2GO documentation (AN12691).

6.2.4 OpenSSL engine/provider configuration

When using OpenSSL as library an abstraction of the OpenSSL cryptographic APIs to the Se05x secure element is required: an example is the signature calculation used for authentication to the EdgeLock 2GO server done via a private key preprovisioned in the Secure Element. To achieve the abstraction the following SSS libraries are used:

- OpenSSL 1.1.1: sss_engine
- OpenSSL 3.x: sssProvider

For OpenSSL 1.1.1 version the sss_engine is used for accessing Se05x functionality:

- Build the sss_engine library together with the el2go_agent application
- Add the path to the built library (libsss_engine.so) in the OpenSSL configuration file which is located under

<SE05X_root_folder>/simw-top/nxp_iot_agent/ex/src/openssl_conf_v111.cnf in the dynamic_path variable of e4sss_section

- Add the environment variable OPENSSL_CONF pointing to the changed openssl_conf_v111.cnf

For OpenSSL 3.x version the sssProvider is used for accessing Se05x functionality. The provider library is linked to the el2go_agent application and is loaded during runtime using the OpenSSL feature of built-in provider support. Note: the sssProvider is always built and installed as a dynamic library, building as static library is not supported. For this reason it is recommended to build the el2go_agent with the option of using shared libraries (-DWithSharedLIB=ON): the binary will be able to dynamically load the sssProvider from the default library installation directory. The use case where the el2go_agent is built with usage of static libraries (-DWithSharedLIB=OFF) will still work, but the binary will expect the sssProvider dynamic library at the same place where it was present during the linking process.

6.2.5 Re-provisioning of objects

If during provisioning of one object the EdgeLock 2GO server finds an objects with same ID, will first delete the object before provision the new one. Now, the following consideration needs to be taken in place when deleting one object:

- the server doesn't know the DELETE permissions of the object on the device (which are defined through policies)
- the server can't read the policies using read attribute APDU since the READ permissions might be disabled

To solve the issue, the server executes two DELETE operation, one unauthenticated and one authenticated, which covers most of possible object status on the device. Now, imagine that you have one object where the unauthenticated DELETE is not allowed while the authenticated is, the first APDU will fail with the 6986 status word. The server will ignore the error returns status of the APDU and continue with provisioning, but on device a warning will be displayed as can be seen in the log below. User can ignore this or similar warnings in case the final report shows success:

```
sss :WARN :APDU Transaction Error: Command not allowed - access denied based on  
object policy (0x6986)
```

6.3 Datastore / Keystore

For storage of credentials and configuration data two types of storage entities are available. A keystore is used for storing sensitive information, typically private keys for a client authentication, whereas a datastore is used for storing configuration data required for connecting to a cloud service. Both are managed remotely from the EdgeLock 2GO cloud service. From the point of view of the EdgeLock 2GO cloud service datastores and keystores are considered endpoints. The EdgeLock 2GO cloud service sends messages to endpoints to set them up according to the desired configuration.

After the device is configured/provisioned for a cloud service by the EdgeLock 2GO cloud service, the relevant information can be extracted for usage in client software from the storages. The access to the credentials is abstracted by using the [SSS APIs](#), configuration data is accessed using a service descriptor struct object.

One keystore implementation is included for supporting the SE050. The EdgeLock 2GO cloud service uses a direct APDU channel to read out from and insert objects into the secure element.

For the sake of demonstration, also two datastore implementations are part of this package. A filesystem based datastore which uses files for storing the data delivered by the EdgeLock 2GO cloud service is present in `<SE05X_root_folder>/simw-top/nxp_iot_agent/*/nxp_iot_agent_datastore_fs.*` (* stands for inc or src folder in the path and for h or c in the file name extension), one that uses raw memory can be found in `<SE05X_root_folder>/simw-top/nxp_iot_agent/*/nxp_iot_agent_datastore_plain.*`.

When writing contents to a datastore, EdgeLock 2GO cloud service protects the data with a checksum. This allows the EdgeLock 2GO agent to check whether the data that is found inside a datastore is valid/uncorrupted.

6.4 Connection to the EdgeLock 2GO cloud service

This section gives a short overview of the communication channel between the EdgeLock 2GO agent and the EdgeLock 2GO cloud service. The connection to the EdgeLock 2GO cloud service is always initiated from the EdgeLock 2GO agent.

6.4.1 Transport layer security

Communication between client and server is protected in a mutually authenticated TLS channel. The TLS protocol versions TLS 1.2 and TLS 1.3 are supported. The supported ciphersuites are:

For TLS 1.2:

- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_DHE_RSA_WITH_AES_256_GCM_SHA384

For TLS 1.3:

- TLS_AES_128_GCM_SHA256
- TLS_AES_256_GCM_SHA384

6.4.2 Client authentication

When using SE050 for authenticating at the EdgeLock 2GO cloud service, the client's private key as well as the client certificate are stored on the secure element. SE050 comes with those credentials already pre-installed from the NXP production site with predefined object identifiers.

There are two crypto libraries available to do the TLS handshake in combination with the SE050. It is possible to use OpenSSL with a custom crypto engine (see [Introduction on OpenSSL engine](#)). Alternatively mbedTLS with an alternative implementation for the SE050 can be used (see [Introduction on MbedTLS ALT \(2.x\) Implementation](#)).

6.4.3 Server authentication

The server is authenticated by using a certificate chain ultimately signed by an NXP root CA. There are two different certificate chains available, one using ECC with the NIST P-384 curve, the other chain uses RSA with 4096 bit keys. The trusted root CA certificates are included with the distributed package of the NXP Plug & Trust Middleware (see also [Parameters for the connection to EdgeLock 2GO cloud service](#)).

The EdgeLock 2GO cloud service provides certificate revocation lists (CRLs) for the CA signing the server certificates. The CRLs are transferred via TLS channel in order to avoid having to implement another protocol (typically http) for retrieving the CRL. When using openssl as crypto library, the CRL processing is skipped for openssl versions < 1.1.1.

6.4.4 Application layer protocol

On the application layer, the EdgeLock 2GO cloud service sends protobuf messages (requests) to individual endpoints which are handled by those. Depending on the endpoint type, different requests are used. Requests to the EdgeLock 2GO agent itself are used for querying the presence of endpoints and their supported features and managing the communication channel. Other requests directly address reading data or writing contents of keystores and datastores.

For configuring an SE050 keystore, the EdgeLock 2GO cloud service uses APDU commands that are directly forwarded to the secure element. If sensitive information is included or integrity protection is required, APDUs can be encrypted. This way a secure end-to-end channel between the EdgeLock 2GO cloud service and the secure element can be established.

For datastores the EdgeLock 2GO cloud service is able to perform read operations to retrieve the current contents. Should it be necessary, an update of the datastore contents can be performed. The EdgeLock 2GO cloud service always replaces the complete contents of the datastore. The first request is an allocate operation, allowing the datastore to make sure memory for the contents is available. It is followed by one or more write operations. If the datastore supports transactions, after the last write, an additional commit operation is done to trigger an atomic update of the datastore contents.

The definition of the protobuf application layer protocol can be found in `<SE05X_root_folder>/simw-top/nxp_iot_agent/doc/protobuf`.

6.4.5 Parameters for the connection to EdgeLock 2GO cloud service

The EdgeLock 2GO agent attempts to take hostname, port, a reference to the client key and client certificate as well as a collection of trusted root ca certificates from a datastore that is registered with a particular id. If a datastore with this id is registered and contains valid data (checksum verification), then the EdgeLock 2GO agent uses its contents. If this is not the case, it falls back to compile-time constants defined in `<SE05X_root_folder>/simw-top/nxp_iot_agent/inc/nxp_iot_agent_config.h`.

For demonstration purposes, in the demo application in `<SE05X_root_folder>/simw-top/nxp_iot_agent/ex/src/iot_agent_demo.c`, a datastore for the EdgeLock 2GO cloud service connection parameters is registered. It is filled at the first boot with the compile-time constants from the configuration file.

In order to be able to mitigate a potential corruption of the keys of the trusted root certificates, in case the connection parameters are taken from the datastore, the EdgeLock 2GO cloud service has the opportunity to update the connection parameters remotely.

6.5 Claim Codes

A claim code allows registering the device into the user account automatically. Claim codes are created and managed from the EdgeLock 2GO service. Please refer to the EdgeLock 2GO documentation (AN12691, section 5.3: 'Add a device to the allowlist using claim codes') for more details.

To facilitate injection of claim code into device, a simple application capable of injecting and deleting claim codes (`el2go_claimcode_inject`) is delivered in combination with the EdgeLock 2GO agent. This application reads a claim code from a text file.

After the claim code was generated on EdgeLock 2GO service, the user has to create a .txt file (`claim.txt` for example) and copy the generated claim code value inside the file. Then, to inject the claim code copied in the file `claim.txt`, the following command can be used:

```
./el2go_claimcode_inject claim.txt
```

Application also supports deleting existing claim code from with the following command:

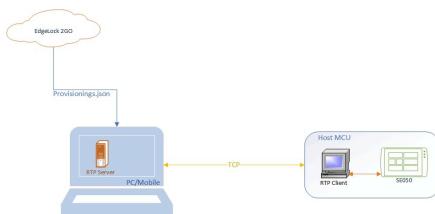
```
./el2go_claimcode_inject --delete
```

6.6 Offline Provisioning of Secure Objects

The EdgeLock 2GO agent supports managed provisioning of secure objects via secure TLS channel (see [Connection to the EdgeLock 2GO cloud service](#)) between device and EdgeLock 2GO. EdgeLock 2GO also supports provisioning of secure objects without a connection from device to EdgeLock 2GO (referred to as offline remote trust provisioning). Please refer to provisioning of secure objects in the EdgeLock 2GO documentation (AN12691, section 8.3: 'Offline secure object provisioning') for more details.

To demonstrate offline remote trust provisioning, a simple client-server example capable of importing secure objects into device is delivered in combination with the EdgeLock 2GO agent. Communication between server-client is implemented by a simple TCP protocol. Below picture depicts a block diagram for offline remote trust provisioning.

- **Block diagram:**



6.6.1 Offline Remote Trust Provisioning Server (RTP Server)

After configuring device and secure objects in your EdgeLock 2GO account, you have the possibility to download provisionings for the device in the form of JSON file. For more details with regard to this step, please refer to EdgeLock 2GO documentation (AN12691, section 8.3: 'Offline secure object provisioning'). The RTP Server application is meant to run on machine capable of connecting to EL2GO and retrieving JSON files containing provisionings. For the sake of simplicity, the RTP Server is implemented in Java language with minimal dependencies and source code is located under:

```
<SE05X_root_folder>/simw-top/nxp_iot_agent/ex/tools/edge-lock-device-link-rtp-server
```

Once the JSON file containing provisionings is downloaded from EL2GO, following commands can be used to build and run application. Please note, it is expected to have maven installed on the machine.

Compile and create jar file:

```
mvn package
```

Print usage details of RTP Server:

```
java -jar target/RtpServer.jar -h
```

Print version details of RTP Server:

```
java -jar target/RtpServer.jar -v
```

Run RTP Server on specified port reading JSON files from specified directory:

```
java -jar target/RtpServer.jar -d c:\el2go -p 7080
```

6.6.2 Offline Remote Trust Provisioning Client (RTP Client)

The RTP Client application is meant to run on the MCU to which the secure element is connected. The build instructions for the RTP Client are similar to that of EdgeLock 2GO agent. The RTP Client application is implemented in C language and source code is located under:

```
<SE05X_root_folder>/simw-top/nxp_iot_agent/ex/apps/remote_provisioning_client.c
```

To start the RTP Client application, the following command can be used:

```
./remote_provisioning_client.exe hostname port where:
```

- hostname = Hostname/IP address of machine on which RTP server is running
- port = Port on which RTP Server is listening

Once the RTP Client is connected, the RTP Server reads the UID of the secure element. The RTP Server parses all JSON files located at the given directory and finds all provisionings for this particular UID. These provisioning are then sent to RTP Client and imported to secure element.

6.7 API

6.7.1 EdgeLock 2GO Main

groupedgelock2go_agent_main

Main API for registering keystores, datastores and connecting to the cloud to update provisionings.

Defines

CLAIMCODE_OBJ_ID

The keyid on the PSA API to use for fetching a claimcode.

EDGELOCK2GO_KEYSTORE_ID

The keystore that stores the credentials for the EdgeLock 2GO cloud service.

When connecting to the EdgeLock 2GO cloud service, for the client authentication a private key is required. This key is expected to be in a keystore which is registered to the EdgeLock 2GO agent.

In case there is a datastore which holds the information how to connect to the EdgeLock 2GO cloud service this datastore also holds the information where to get the private key.

In case the EdgeLock 2GO agent needs to fall back to compile-time constant connection information, it does assume that the private key for the client authentication is stored in a keystore that is registered with this ID.

TypeDefs

typedef struct _nxp_iot_UpdateStatusReport **nxp_iot_UpdateStatusReport**

typedef struct pb_field_s **pb_field_t**

typedef struct pb_istream_s **pb_istream_t**

typedef struct pb_ostream_s **pb_ostream_t**

Functions

void **iot_agent_free_service_descriptor**(nxp_iot_ServiceDescriptor *service_descriptor)

Free all FT_POINTER fields of a service descriptor.

When selecting a service, a service descriptor is read from a datastore. A service descriptor can contain fields of variable length (binary data (certificates, etc.) or text (hostname, etc.)). Those fields use dynamically allocated memory. The memory is freed by calling this function.

Parameters

service_descriptor – [in] Reference to service descriptor

void **iot_agent_free_update_status_report**(nxp_iot_UpdateStatusReport *status_report)

Free all FT_POINTER fields of a update status report.

When a status report is filled during updating a device configuration, it contains pointer fields which use dynamically allocated memory. The memory is freed by calling this function.

Parameters

status_report – [in] Reference to status_report

iot_agent_status_t **iot_agent_get_datastore_by_id**(const **iot_agent_context_t** *ctx, const uint32_t id, **iot_agent_datastore_t** **datastore)

Get a reference to a datastore based on its identifier.

Parameters

- **ctx** – [in] The context of the agent.
- **id** – [in] The datastore ID.
- **datastore** – [out] The output datastore.

Return values

IOT_STATUS_SUCCESS – A datastore with the `id` was found in the agent's context and a pointer to it is returned in `datastore`.

```
iot_agent_status_t iot_agent_get_datastore_index_by_id(const iot_agent_context_t *ctx, const uint32_t id, size_t *index)
```

Get the index of a datastore based on its identifier.

Parameters

- `ctx` – **[in]** The context of the agent.
- `id` – **[in]** The datastore ID.
- `index` – **[out]** The output index.

Return values

IOT_STATUS_SUCCESS – A datastore with the `id` was found in the agent's context and its index is returned in `index`.

```
bool iot_agent_get_endpoint_info(void *context, void *endpoint_information)
```

Get an endpoint information of the endpoint.

Parameters

- `context` – **[in]** Reference to end point context
- `endpoint_information` – **[in]** Reference to end point information

```
iot_agent_status_t iot_agent_get_keystore_by_id(const iot_agent_context_t *ctx, const uint32_t id, iot_agent_keystore_t **keystore)
```

Get a reference to a keystore based on its identifier.

Parameters

- `ctx` – **[in]** The context of the agent.
- `id` – **[in]** The datastore ID.
- `keystore` – **[out]** The output keystore.

Return values

IOT_STATUS_SUCCESS – A keystore with the `id` was found in the agent's context and a pointer to it is returned in `keystore`.

```
iot_agent_status_t iot_agent_get_keystore_index_by_id(const iot_agent_context_t *ctx, const uint32_t id, size_t *index)
```

Get the index of a keystore based in its identifier.

Parameters

- `ctx` – **[in]** The context of the agent.
- `id` – **[in]** The keystore ID.
- `index` – **[out]** The output index.

Return values

IOT_STATUS_SUCCESS – A keystore with the `id` was found in the agent's context and its index is returned in `index`.

```
size_t iot_agent_get_number_of_services(const iot_agent_context_t *ctx)
```

Returns total number of services of all registered datastores.

Parameters

- `ctx` – **[in]** Context for the `iot_agent`.

Return values

Total – number of services of all registered datastores

```
iot_agent_status_t iot_agent_get_service_descriptor(const iot_agent_context_t *ctx, nxp_iot_ServiceDescriptor *service_descriptor)
```

Get the service descriptor of the currently selected service.

Parameters

- **ctx** – **[in]** The context of the agent.
- **service_descriptor** – **[out]** Structure for holding a service descriptor. Must point to a valid service descriptor object upon invocation. Any FT_POINTER fields in the service descriptor are freed before changing the contents to the service_descriptor of the selected service.

Return values

IOT_AGENT_SUCCESS – Upon success

Pre

The configuration data associated to the context is valid.

Post

In case of success, the service_descriptor structure is filled and needs to be freed after usage by calling [iot_agent_free_service_descriptor](#). In case of failure no freeing is required. Also in case of failures, the contents of service_descriptor are not guaranteed to remain intact.

```
bool iot_agent_handle_request(pb_istream_t *istream, pb_ostream_t *ostream, const pb_field_t *message_type, void *context)
```

handle request by end point

Parameters

- **istream** – **[in]** Input stream
- **ostream** – **[in]** Output stream
- **message_type** – **[in]** a pointer to the message type fields array
- **context** – **[in]** End point context

```
iot_agent_status_t iot_agent_init(iot_agent_context_t *ctx)
```

Initialize EdgeLock 2GO agent context memory with zeros.

Parameters

ctx – **[inout]** EdgeLock 2GO agent context

Return values

IOT_AGENT_SUCCESS – Upon success

```
iot_agent_status_t iot_agent_init_dispatcher(iot_agent_dispatcher_context_t *dispatcher_context, iot_agent_context_t *agent_context, nxp_iot_ServiceDescriptor *service_descriptor, nxp_iot_UpdateStatusReport *status_report)
```

Initialize Dispatcher.

Parameters

- **dispatcher_context** – [in] Context for the dispatcher
- **agent_context** – [in] Context for the agent
- **service_descriptor** – [in] The service descriptor containing the connection parameters to connect to the EdgeLock 2GO cloud service.
- **status_report** – [out] A pointer to a structure that gets filled with a status report after the update is complete. If NULL is given, no status report is created.

`bool iot_agent_is_service_configuration_data_valid(const iot_agent_context_t *ctx)`

Checks whether service configuration data of all registered datastores is valid.

Parameters

`ctx` – [in] Context for the iot_agent.

Return values

- **true** – Service configuration data of all registered datastores is valid
- **false** – Service configuration data of a registered datastores is invalid

`void iot_agent_print_status_report(const nxp_iot_UpdateStatusReport *status_report)`

Print status report.

It prints the status report of the agent execution.

Parameters

`status_report` – [in] Reference to status_report

`iot_agent_status_t iot_agent_register_datastore(iot_agent_context_t *ctx, iot_agent_datastore_t *datastore)`

Register datastore endpoint.

Note that the ownership for the datastore is not transferred. The caller is responsible that the datastore is freed at the appropriate time.

It is not possible to register two endpoints with the same identifier.

Parameters

- `ctx` – [in] Context for the iot_agent.
- `datastore` – [in] Datastore that is registered.

Return values

`IOT_AGENT_SUCCESS` – Upon success

`iot_agent_status_t iot_agent_register_keystore(iot_agent_context_t *ctx, iot_agent_keystore_t *keystore)`

Register a keystore endpoint.

Note that the ownership for the keystore is not transferred. The caller is responsible that the keystore is freed at the appropriate time.

It is not possible to register two endpoints with the same identifier.

Parameters

- `ctx` – Context for the iot_agent.
- `keystore` – [in] Keystore that is registered.

Return values

`IOT_AGENT_SUCCESS` – Upon success

`iot_agent_status_t iot_agent_select_service_by_id(iot_agent_context_t *ctx, uint64_t service_id, nxp_iot_ServiceDescriptor *service_descriptor)`

Select service by given ID.

See Also:

[iot_agent_is_service_configuration_data_valid](#)**Parameters**

- **ctx** – **[in]** Context for the iot_agent.
- **service_id** – **[in]** ID of the service
- **service_descriptor** – **[out]** Structure for holding a service descriptor. Must point to a valid service descriptor object upon invocation. Any FT_POINTER fields in the service descriptor are freed before changing the contents to the service_descriptor of the selected service.

Return values

IOT_AGENT_SUCCESS – Upon success

Pre

The configuration data associated to the context is valid.

Post

In case of success, the service_descriptor structure is filled and needs to be freed after usage by calling [iot_agent_free_service_descriptor](#). In case of failure no freeing is required. Also in case of failures, the contents of service_descriptor are not guaranteed to remain intact.

iot_agent_status_t **iot_agent_select_service_by_index**([iot_agent_context_t](#) *ctx, size_t index, [nxp_iot_ServiceDescriptor](#) *service_descriptor)

Select service by given index.

See Also:[iot_agent_is_service_configuration_data_valid](#)**Parameters**

- **ctx** – **[in]** Context for the iot_agent.
- **index** – **[in]** Index of the service
- **service_descriptor** – **[out]** Structure for holding a service descriptor. Must point to a valid service descriptor object upon invocation. Any FT_POINTER fields in the service descriptor are freed before changing the contents to the service_descriptor of the selected service.

Return values

IOT_AGENT_SUCCESS – Upon success

Pre

The configuration data associated to the context is valid.

Post

In case of success, the service_descriptor structure is filled and needs to be freed after usage by calling [iot_agent_free_service_descriptor](#). In case of failure no freeing is required. Also in case of failures, the contents of service_descriptor are not guaranteed to remain intact.

iot_agent_status_t **iot_agent_set_edgelock2go_datastore**([iot_agent_context_t](#) *ctx, [iot_agent_datastore_t](#) *datastore)

Set the datastore that is used to hold the information to connect to the EdgeLock 2GO cloud service.

Note that the ownership for the datastore is not transferred. The caller is responsible that the datastore is freed at the appropriate time.

Parameters

- **ctx** – [in] Context for the iot_agent.
- **datastore** – [in] Datastore to be set.

`iot_agent_status_t iot_agent_update_device_configuration(iot_agent_context_t *agent_context, nxp_iot_UpdateStatusReport *status_report)`

Update device configuration Reach out to EdgeLock 2GO cloud service for checking and (if applicable) fetching configuration updates for the device.

Parameters

- **agent_context** – [in] Context for the iot_agent.
- **status_report** – [out] Provides a more detailed view on the operations performed during the update and its outcomes. If the argument is NULL, no detailed status is reported.

Return values

- **IOT_AGENT_SUCCESS** – Upon success
- **IOT_AGENT_FAILURE** – Upon failure

Post

In case of success, the status_report structure is filled using dynamically allocated fields and needs to be freed after usage by calling #iot_agent_free_status_report.

`iot_agent_status_t iot_agent_update_device_configuration_from_constants(iot_agent_context_t *agent_context, nxp_iot_UpdateStatusReport *status_report)`

Update device configuration from constants.

Reach out to EdgeLock 2GO cloud service for checking and (if applicable) fetching configuration updates for the device.

The connection details (hostname/port/server root certificates, etc.) are taken from the configuration constants in nxp_iot_agent_config.h.

It is necessary that an sss keystore that contains credentials (client key and client certificate) for connecting to the EdgeLock 2GO cloud service. The object ids to those credentials are settable via function arguments.

Parameters

- **ctx** – [in] Context for the iot_agent.
- **status_report** – [out] Provides a more detailed view on the operations performed during the update and its outcomes. If the argument is NULL, no detailed status is reported.

Return values

- **IOT_AGENT_SUCCESS** – Upon success
- **IOT_AGENT_FAILURE** – Upon failure

Post

In case of success, the status_report structure is filled using dynamically allocated fields and needs to be freed after usage by calling #iot_agent_free_status_report.

`iot_agent_status_t iot_agent_update_device_configuration_from_datastore(iot_agent_context_t *agent_context, iot_agent_datastore_t *datastore, nxp_iot_UpdateStatusReport *status_report)`

Update device configuration from datastore.

Reach out to EdgeLock 2GO cloud service for checking and (if applicable) fetching configuration updates for the device.

The connection details (hostname/port/server root certificates, etc.) are taken from the datastore passed as input parameter

It is necessary that an sss keystore that contains credentials (client key and client certificate) for connecting to the EdgeLock 2GO cloud service. The object ids to those credentials are settable via function arguments.

Parameters

- **ctx – [in]** Context for the iot_agent.
- **datastore – [in]** Datastore including the connection parameters
- **status_report – [out]** Provides a more detailed view on the operations performed during the update and its outcomes. If the argument is NULL, no detailed status is reported.

Return values

- **IOT_AGENT_SUCCESS** – Upon success
- **IOT_AGENT_FAILURE** – Upon failure

Post

In case of success, the status_report structure is filled using dynamically allocated fields and needs to be freed after usage by calling #iot_agent_free_status_report.

`iot_agent_status_t iot_agent_update_device_configuration_from_service_descriptor(iot_agent_context_t *agent_context, nxp_iot_ServiceDescriptor *service_descriptor, nxp_iot_UpdateStatusReport *status_report)`

Update device configuration from service descriptor.

Reach out to EdgeLock 2GO cloud service for checking and (if applicable) fetching configuration updates for the device.

The connection details (hostname/port/server root certificates, etc.) are taken from the service descriptor passed as input parameter

It is necessary that an sss keystore that contains credentials (client key and client certificate) for connecting to the EdgeLock 2GO cloud service. The object ids to those credentials are settable via function arguments.

Parameters

- **ctx – [in]** Context for the iot_agent.
- **service_descriptor – [in]** Service descriptor including the connection parameters
- **status_report – [out]** Provides a more detailed view on the operations performed during the update and its outcomes. If the argument is NULL, no detailed status is reported.

Return values

- **IOT_AGENT_SUCCESS** – Upon success
- **IOT_AGENT_FAILURE** – Upon failure

Post

In case of success, the status_report structure is filled using dynamically allocated fields and needs to be freed after usage by calling #iot_agent_free_status_report.

6.7.2 EdgeLock 2GO Datastore

groupedgelock2go_agent_datastore

Functions that define how to interact with a datastore. There exist multiple concrete implementations for different microcontrollers.

TypeDefs

```
typedef iot_agent_status_t (*iot_agent_datastore_allocator_t)(void *context, size_t len)
typedef iot_agent_status_t (*iot_agent_datastore_committer_t)(void *context)
typedef iot_agent_status_t (*iot_agent_datastore_destroyer_t)(void *context)
typedef iot_agent_status_t (*iot_agent_datastore_reader_t)(void *context, void *dst, size_t offset, size_t *len)
typedef bool (*iot_agent_datastore_validator_t)(const iot_agent_datastore_t *context)
typedef iot_agent_status_t (*iot_agent_datastore_writer_t)(void *context, size_t offset, const void *src, size_t len)
typedef struct _nxp_iot_ResponsePayload nxp_iot_ResponsePayload
```

Functions

bool **datastore_read_callback**(*pb_istream_t* *stream, *uint8_t* *buf, *size_t* count)

iot_agent_status_t **iot_agent_datastore_allocate**(*iot_agent_datastore_t* *datastore, *size_t* len)

Allocate memory in a datastore.

Reserve memory in a datastore that can be written to by subsequent writes.

If the memory allocation fails, this function does not return IOT_AGENT_SUCCESS.

iot_agent_status_t **iot_agent_datastore_commit**(*iot_agent_datastore_t* *datastore)

Commit a transaction to a datastore.

A call to this function indicates that all necessary writes of a transaction are done and the datastore contents can be considered valid.

This is the indication to the datastore to invalidate old contents and from point in time onwards use the data that was updated with the transaction that is committed.

bool **iot_agent_datastore_encode_datastore_ok_response**(*pb_ostream_t* *ostream)

iot_agent_status_t **iot_agent_datastore_free**(*iot_agent_datastore_t* *datastore)

Destroy the datastore.

Depending on the type of the datastore this is triggering actions on the datastore's context itself by calling the *_destroy()* function of the datastore interface.

This does not free the context of the datastore.

The datastore is not usable after a call to *iot_agent_datastore_free*.

iot_agent_status_t **iot_agent_datastore_init**(*iot_agent_datastore_t* *datastore, *uint32_t* identifier, *const char *basename*, *iot_agent_datastore_validator_t* validator)

Initialize a Agent datastore.

The Agent datastore; internally executes all the necessary operations which allows the specific datastore to be initialized

iot_agent_status_t **iot_agent_datastore_read**(*iot_agent_datastore_t* *datastore, *void* *dst, *size_t* offset, *size_t* *len)

Read from a datastore to a buffer in memory.

Starting at position *offset*, *len* bytes are read from the datastore and copied to the memory pointed to by *dst*. If the datastore does not hold *len* bytes or the read would be out of bounds, only as many bytes as available are read.

len is updated to hold the number of bytes that actually were read from the datastore.

No length checks on *dst* are performed, the caller is responsible that the memory location is writeable and can hold *len* bytes.

iot_agent_status_t **iot_agent_datastore_write**(*iot_agent_datastore_t* *datastore, *size_t* offset, *const void* *src, *size_t* len)

Write from a buffer in memory to a datastore.

From *src*, *len* bytes are copied to the datastore. The first byte is written to the position *offset* in the datastore.

Length check on the destination - the datastore - is performed, memory in the datastore needs to be pre-allocated. If the data does not fit, this function does not return IOT_AGENT_SUCCESS.

struct **datastore_stream_context_t**

#include <nxp_iot_agent_datastore.h>

A stream for reading contents from within a datastore, starting at an offset.

struct **iot_agent_datastore_interface_t**

#include <nxp_iot_agent_datastore.h>

The interface any datastore needs to implement.

struct **iot_agent_datastore_t**

#include <nxp_iot_agent_datastore.h>

A context holding the state of a datastore; this is passed to datastore interface functions.

6.7.3 EdgeLock 2GO Keystore

groupededge2go_keystore

Functions to interact with a keystore. There are concrete implementations of this API for different SEs.

TypeDefs

typedef iot_agent_status_t (***iot_agent_keystore_destroyer_t**)(void *context)

typedef void (***iot_agent_keystore_session_closer_t**)(void *context)

typedef iot_agent_status_t (***iot_agent_keystore_session_opener_t**)(void *context)

Functions

void **iot_agent_keystore_close_session**(*iot_agent_keystore_t* *keystore)

Close a session/connection to a keystore.

iot_agent_status_t **iot_agent_keystore_free**(*iot_agent_keystore_t* *keystore)

Destroy the keystore.

Depending on the type of the keystore this is triggering actions on the keystore's context itself by calling the `_destroy()` function of the keystore interface.

The `keystore` is not usable after a call to `iot_agent_keystore_free`.

iot_agent_status_t **iot_agent_keystore_init**(*iot_agent_keystore_t* *keystore, uint32_t identifier, *iot_agent_platform_context_t* *platform_context)

Initialize an Agent keystore.

The Agent keystore; internally executes all the necessary operations which allows the specific keystore to be initialized as for example opening a SE5x session with the secure element if necessary

If the function does not return IOT_AGENT_SUCCESS, it can be assumed that no memory was allocated. The datastore is unusable.

iot_agent_status_t **iot_agent_keystore_open_session**(*iot_agent_keystore_t* *keystore)

Open a session/connection to a keystore.

struct **iot_agent_keystore_interface_t**

#include <nxp_iot_agent_keystore.h>

The interface any keystore needs to implement.

struct **iot_agent_keystore_key_ref_t**

#include <nxp_iot_agent_keystore.h>

A structure used to abstract key reference between different OpenSSL versions.

struct **iot_agent_keystore_t**

#include <nxp_iot_agent_keystore.h>

A structure binding a keystore interface and a keystore context to a keystore instance.

6.7.4 EdgeLock 2GO Session

Warning:

doxygengroup: Cannot find group “edgelock2go_agent_session” in doxygen xml output for project “simw” from directory: ../../_docs_build_simw/xml

6.7.5 EdgeLock 2GO Service

groupedgelock2go_agent_service

Functionality to work with service descriptors. A service descriptor represents all information from a single cloud provisioning.

Defines

IOT_AGENT_CONFIGURATION_DATA_VERSION

TypeDefs

typedef __PACKED_STRUCT **configuration_data_header_t**

The header of configuration data stored in a datastore.

typedef uint8_t **public_key_identifier_t**[16]

typedef uint8_t **service_identifier_t**[32]

Functions

void **iot_agent_service_free_service_descriptor**(nxp_iot_ServiceDescriptor *service_descriptor)

Free all FT_POINTER fields of a service descriptor.

When selecting a service, a service descriptor is read from a datastore. A service descriptor can contain fields of variable length (binary data (certificates, etc.) or text (hostname, etc.)). Those fields use dynamically allocated memory. The memory is freed by calling this function.

Parameters

service_descriptor – [in] Reference to service descriptor

size_t **iot_agent_service_get_number_of_services**(const **iot_agent_datastore_t** *ctx)

Get the number of services that are available.

Parameters

ctx – [in] Context for the iot_agent.

Returns

The number of services that are available.

iot_agent_status_t iot_agent_service_get_protocol_of_service_as_string(const nxp_iot_ServiceDescriptor *service_descriptor, const char **buffer)

Get a textual description of the protocol.

Protocol strings are internal constant c-strings. This function returns a pointer to such a string, ownership remains with the iot_agent.

Parameters

- **service_descriptor – [in]** The service descriptor to get the protocol for.
- **buffer – [out]** A pointer to a c-string that is changed to point to the textual representation of the protocol.

Return values

- **IOT_AGENT_SUCCESS** – The service type could be resolved.
- **IOT_AGENT_FAILURE** – The service type is invalid.

```
iot_agent_status_t iot_agent_service_get_service_descriptor_of_service(const iot_agent_datastore_t *datastore,  
size_t offset, nxp_iot_ServiceDescriptor *service_descriptor)
```

Get the service descriptor of a service specified by offset.

See Also:

[iot_agent_is_service_configuration_data_valid](#)

Parameters

- **ctx – [in]** The datastore to query for the service.
- **offset – [in]** The offset of the service - the memory location in the given datastore.
- **service_descriptor – [out]** Structure for holding a service descriptor. Must point to a valid service descriptor object upon invocation. Any FT_POINTER fields in the service descriptor are freed before changing the contents to the service_descriptor of the selected service.

Return values

IOT_AGENT_SUCCESS – Upon success

Pre

The configuration data associated to the context is valid.

Post

In case of success, the service_descriptor structure is filled and needs to be freed after usage by calling [iot_agent_free_service_descriptor](#). In case of failure no freeing is required. Also in case of failures, the contents of service_descriptor are not guaranteed to remain intact.

```
iot_agent_status_t iot_agent_service_get_service_offset_by_id(const iot_agent_datastore_t *datastore, uint64_t  
service_id, size_t *offset, nxp_iot_ServiceDescriptor *service_descriptor)
```

Get the offset and the service descriptor of a service specified by service_id.

See Also:

[iot_agent_is_service_configuration_data_valid](#)

Parameters

- **datastore – [in]** The datastore to query for the service.
- **service_id – [in]** ID of the service
- **offset – [out]** The offset of the service descriptor within the datastore.
- **service_descriptor – [out]** Structure for holding a service descriptor. Must point to a valid service descriptor object upon invocation. Any FT_POINTER fields in the service descriptor are freed before changing the contents to the service_descriptor of the selected service.

Return values

IOT_AGENT_SUCCESS – Upon success

Pre

The configuration data associated to the context is valid.

Post

In case of success, the service_descriptor structure is filled and needs to be freed after usage by calling [iot_agent_free_service_descriptor](#). In case of failure no freeing is required. Also in case of failures, the contents of service_descriptor are not guaranteed to remain intact.

`iot_agent_status_t iot_agent_service_get_service_offset_by_index(const iot_agent_datastore_t *datastore, size_t index, size_t *offset, nxp_iot_ServiceDescriptor *service_descriptor)`

Get the offset and the service descriptor of a service specified by index.

See Also:

[iot_agent_is_service_configuration_data_valid](#)

Parameters

- **ctx – [in]** The datastore to query for the service.
- **index – [in]** The index of the service (within the given datastore).
- **service_descriptor – [out]** Structure for holding a service descriptor. Must point to a valid service descriptor object upon invocation. Any FT_POINTER fields in the service descriptor are freed before changing the contents to the service_descriptor of the selected service.

Return values

IOT_AGENT_SUCCESS – Upon success

Pre

The configuration data associated to the context is valid.

Post

In case of success, the service_descriptor structure is filled and needs to be freed after usage by calling [iot_agent_free_service_descriptor](#). In case of failure no freeing is required. Also in case of failures, the contents of service_descriptor are not guaranteed to remain intact.

`iot_agent_status_t iot_agent_service_get_service_type_as_string(const nxp_iot_ServiceDescriptor *service_descriptor, const char **buffer)`

Get a textual description of the service type.

Service type strings are internal constant c-strings. This function returns a pointer to such a string, ownership remains with the iot_agent.

Parameters

- **service_descriptor – [in]** The service descriptor to get the type for.
- **buffer – [out]** A pointer to a c-string that is changed to point to the textual representation of the service type.

Return values

- **IOT_AGENT_SUCCESS** – The service type could be resolved.
- **IOT_AGENT_FAILURE** – The service type is invalid.

`bool iot_agent_service_is_configuration_data_valid(const iot_agent_datastore_t *ctx)`

Checks service configuration data of all registered datastores are valid.

Parameters

ctx – [in] Context for the iot_agent.

Return values

- **true** – Service configuration data of all registered datastores are valid
- **false** – Service configuration data of a registered datastores is invalid

Variables**uint32_t length**

Total length of the stored data (incl. all fields of configuration_data_header_t).

uint32_t number_of_services

Number of stored service configurations.

uint32_t version

Version number of the structure of the configuration data.

struct service_link_t

6.7.6 EdgeLock 2GO Dispatcher

groupedgelock2go_agent_dispatcher

The dispatcher handles and decodes requests of endpoints.

TypeDefs**typedef bool (*get_endpoint_info_callback_t)(void *context, void *endpoint_information)****typedef struct iot_agent_context_t iot_agent_context_t****typedef bool (*request_handler_callback_t)(pb_istream_t *istream, pb_ostream_t *ostream, const pb_field_t *message_type, void *endpoint_context)****Enums****enum _iot_agent_stream_type_t**

Values:

enumerator STREAM_TYPE_NETWORK

Network stream

enumerator STREAM_TYPE_BUFFER_REQUESTS

Buffer stream in Requests format

Functions**bool encode_responses_callback(pb_ostream_t *ostream, const pb_field_t *field, void *const *arg)**

Encodes the responses callback.

Parameters

- **ostream** – **[out]** Output stream
- **field** – **[in]** Input pb field
- **args** – **[in]** Payload arguments

bool encode_responses_from_payload(pb_ostream_t *ostream, nxp_iot_ResponsePayload *response)

Encodes the response from payload.

Parameters

- **ostream** – **[out]** Output stream
- **response** – **[in]** The input payload

```
bool handle_request_payload(pb\_istream\_t *stream, const pb\_field\_t *field, void **arg)
```

Handles the payload request.

Parameters

- **stream** – [in] Input stream
- **filed** – [in] Input pb field
- **args** – [in] Payload arguments

```
bool handle_requests(pb\_istream\_t *stream, const pb\_field\_t *field, void **arg)
```

Handles the request.

Parameters

- **stream** – [in] Input stream
- **filed** – [in] Input pb field
- **args** – [in] Payload arguments

```
iot_agent_status_t iot_agent_dispatcher(iot\_agent\_dispatcher\_context\_t *dispatcher_context, pb\_istream\_t *input, pb\_ostream\_t *output)
```

Dispatcher.

Parameters

- **dispatcher_context** – [in] Context for the dispatcher
- **input** – [in] Input stream
- **out** – [in] Output stream

```
struct handle_request_payload_args_t
```

```
#include <nxp_iot_agent_dispatcher.h>
```

Context structure for passing dispatcher information to callbacks for message decoding.

```
struct iot_agent_dispatcher_context_t
```

```
struct iot_agent_endpoint_cache_table_entry_t
```

6.7.7 EdgeLock 2GO Utils

groupedgelock2go_agent_utils

Utility functions for the EdgeLock 2GO agent for writing and generating key references and certificates.

Defines

```
NXP_IOT_AGENT_EDGELOCK2GO_CLIENT_CERTIFICATE_BUFFER_SIZE
```

The size of the buffer to reserve for the EdgeLock 2GO cloud service client certificate.

Functions

```
iot_agent_status_t iot_agent_get_first_found_object(iot\_agent\_keystore\_t *keystore, uint32_t *object_ids, size_t num_objects, uint32_t *object_id)
```

Maps a given service id to the range of keys that are managed by the EdgeLock 2GO cloud service.

Parameters

- **keystore** – [in] The input keystore
- **object_ids** – [in] The list of object IDs
- **num_objects** – [in] The number of objects
- **object_id** – [out] The output object id

Return values

- **IOT_AGENT_SUCCESS** – upon success
- **IOT_AGENT_FAILURE** – upon failure

`iot_agent_status_t iot_agent_keystore_file_existence(const char *filename, bool forceCreation)`

Checks existence of a file. If required it forces creation of file.

Parameters

- **filename** – **[in]** Name of the file
- **forceCreation** – **[in]** Switch to force creation of the file

Return values

- **IOT_AGENT_SUCCESS** – upon success
- **IOT_AGENT_FAILURE** – upon failure

`iot_agent_status_t iot_agent_pad_iso7816d4(uint8_t *data, size_t data_size, size_t unpadded_length, size_t blocksize, size_t *padded_length)`

Append padding bytes to a given message.

Padding is done in-place, meaning the data buffer has to be big enough to hold also the padding bytes.

Parameters

- **data** – **[inout]** Buffer to be unpad
- **data_size** – **[in]** The length of the buffer
- **unpadded_length** – **[in]** The length of the unpadded part
- **blocksize** – **[in]** Size of the block
- **padded_length** – **[inout]** The length of the padded part

Return values

- **IOT_AGENT_SUCCESS** – upon success
- **IOT_AGENT_FAILURE** – upon failure

`iot_agent_status_t iot_agent_utils_configure_edgelock2go_datastore(iot_agent_keystore_t *keystore, iot_agent_datastore_t *datastore, int argc, const char *argv[])`

Assemble a service descriptor for the connection to EdgeLock 2GO cloud service and write it to a datastore.

The function gets the EDGELOCK2GO_HOSTNAME and EDGELOCK2GO_PORT from different inputs respecting the following priority order:

- Command line arguments
- Environment variables
- Existing valid datastore file
- Macro defines

Both hostname and port must be provided on the input that the function is configure it from there

It is assumed that credentials (either ECC or RSA) for client certificate authentication are stored in `keystore`.

Parameters

- **keystore** – **[in]** The input keystore
- **datastore** – **[out]** The datastore where to write
- **argc** – **[in]** Number of arguments
- **argv** – **[in]** Argument array

Return values

- **IOT_AGENT_SUCCESS** – upon success
- **IOT_AGENT_FAILURE** – upon failure

```
iot_agent_status_t iot_agent_utils_convert_service2key_id(uint64_t service_id, uint32_t *key_id_ptr)
```

Converts the service ID to key id.

Parameters

`service_id` – [in] Service ID @ param[out] `key_id_ptr` Key ID

Return values

- `IOT_AGENT_SUCCESS` – upon success
- `IOT_AGENT_FAILURE` – upon failure

```
iot_agent_status_t iot_agent_utils_create_self_signed_edgelock2go_certificate(iot\_agent\_keystore\_t *keystore,  
uint8_t *certificate_buffer, size_t *certificate_buffer_size)
```

Creates a self signed certificate.

Parameters

- `keystore` – [in] The input keystore
- `certificate_buffer` – [out] Buffer including the certificate
- `certificate_buffer_size` – [inout] the length of the certificate

Return values

- `IOT_AGENT_SUCCESS` – upon success
- `IOT_AGENT_FAILURE` – upon failure

```
iot_agent_status_t iot_agent_utils_gen_key_ref(const iot\_agent\_keystore\_t *keystore, uint32_t key_id, iot\_agent\_keystore\_key\_ref\_t *gen_key_ref)
```

Generates the key reference. The function is used when OpenSSL crypto is selected to generate the Open SSL key reference using the key stored in the keystore. The key reference provided as output depends on the used OpenSSL version.

- Version < 3.x.x: EVP_PKEY
- Version >= 3.x.x: OSSL_PARAM

Parameters

- `keystore` – [in] The input keystore
- `key_id` – [in] ID of the key
- `gen_key_ref` – [out] The generated key reference

Return values

- `IOT_AGENT_SUCCESS` – upon success
- `IOT_AGENT_FAILURE` – upon failure

```
iot_agent_status_t iot_agent_utils_get_certificate_common_name(iot\_agent\_context\_t *ctx, const nxp_iot_ServiceDescriptor *service_descriptor, char *common_name, size_t max_size)
```

Gets the common name from the client certificate.

Parameters

- `ctx` – [in] Context for the iot_agent
- `service_descriptor` – [in] Descriptor with service data
- `common_name` – [inout] Common name string
- `max_size` – [in] Maximum size

Return values

- `IOT_AGENT_SUCCESS` – upon success
- `IOT_AGENT_FAILURE` – upon failure

```
iot_agent_status_t iot_agent_utils_get_certificate_from_keystore(iot\_agent\_keystore\_t *keystore, uint32_t  
certificate_id, uint8_t *cert, size_t *cert_len)
```

Read a certificate from a keystore.

Parameters

- **keystore** – [in] The input keystore
- **certificate_id** – [in] ID of the certificate
- **cert** – [out] Buffer including the certificate
- **cert_len** – [inout] the length of the certificate

Return values

- **IOT_AGENT_SUCCESS** – upon success
- **IOT_AGENT_FAILURE** – upon failure

`iot_agent_status_t iot_agent_utils_get_device_id(uint8_t *buffer, size_t *len)`

Retrieve the UID of a device.

Parameters

- **buffer** – [out] The buffer including the UUID
- **len** – [inout] The length of the buffer

Return values

- **IOT_AGENT_SUCCESS** – upon success
- **IOT_AGENT_FAILURE** – upon failure

`iot_agent_status_t iot_agent_utils_get_edgelock2go_certificate_id(iot_agent_keystore_t *keystore, uint32_t *object_id)`

Checks whether a keystore contains the object with the defined certificate to use for authenticating at the Edge Lock 2GO cloud service.

The keystore might contain keys using ECC and/or RSA. If available, it will return the object id of the ECC certificate, otherwise the object id of the RSA certificate.

When neither is found or in case of other issues with the keystore, the function does not return IOT_AGENT_SUCCESS.

Parameters

- **keystore** – [in] The input keystore
- **object_id** – [out] ID of the certificate

Return values

- **IOT_AGENT_SUCCESS** – upon success
- **IOT_AGENT_FAILURE** – upon failure

`iot_agent_status_t iot_agent_utils_get_edgelock2go_key_id(iot_agent_keystore_t *keystore, uint32_t *object_id)`

Checks whether a keystore contains the object with the defined key to use for authenticating at the EdgeLock 2GO cloud service.

The keystore might contain keys using ECC and/or RSA. If available, it will return the object id of the ECC key, otherwise the object id of the RSA key.

When neither is found or in case of other issues with the keystore, the function does not return IOT_AGENT_SUCCESS.

Parameters

- **keystore** – [in] The input keystore
- **object_id** – [out] ID of the key

Return values

- **IOT_AGENT_SUCCESS** – upon success
- **IOT_AGENT_FAILURE** – upon failure

```
iot_agent_status_t iot_agent_utils_get_keystore_from_service_descriptor(const iot_agent_context_t *ctx, const nxp_iot_ServiceDescriptor *service_descriptor, iot_agent_keystore_t **keystore)
```

Gets the keystore from service descriptor.

Parameters

- **ctx** – [in] Agent context
- **service_descriptor** – [in] Input service descriptor
- **keystore** – [out] The output keystore

Return values

- **IOT_AGENT_SUCCESS** – upon success
- **IOT_AGENT_FAILURE** – upon failure

```
iot_agent_status_t iot_agent_utils_write_certificate_pem(uint8_t *buffer, size_t len, const char *filename)
```

Create a PEM file containing a certificate chain.

Parameters

- **buffer** – [in] Buffer containing the encoded certificate
- **len** – [in] length of the buffer
- **filename** – [in] Name of the PEM file to be created

Return values

- **IOT_AGENT_SUCCESS** – upon success
- **IOT_AGENT_ERROR_CRYPTO_ENGINE_FAILED** – upon crypto operation failure
- **IOT_AGENT_ERROR_FILE_SYSTEM** – upon failure while creating or writing certificate file
- **IOT_AGENT_FAILURE** – upon failure

```
iot_agent_status_t iot_agent_utils_write_certificate_pem_cos_over_rtp(iot_agent_context_t *ctx, const nxp_iot_ServiceDescriptor *service_descriptor, const char *filename)
```

Create a PEM file containing service certificate stored in keystore.

Parameters

- **ctx** – [in] Context for the iot_agent
- **service_descriptor** – [in] Descriptor with service data
- **filename** – [in] Name of the PEM file to be created

Return values

- **IOT_AGENT_SUCCESS** – upon success
- **IOT_AGENT_FAILURE** – upon failure

```
iot_agent_status_t iot_agent_utils_write_certificate_pem_from_keystore(iot_agent_keystore_t *keyStore, const uint32_t certid, const char *filename)
```

Create a PEM file containing a certificate.

Parameters

- **keyStore** – [in] Key store context
- **certid** – [in] Certificate id
- **filename** – [in] Name of the PEM file to be created

Return values

- **IOT_AGENT_SUCCESS** – upon success
- **IOT_AGENT_ERROR_CRYPTO_ENGINE_FAILED** – upon crypto operation failure
- **IOT_AGENT_ERROR_FILE_SYSTEM** – upon failure while creating or writing certificate file
- **IOT_AGENT_FAILURE** – upon failure

`iot_agent_status_t iot_agent_utils_write_edgelock2go_datastore(iot_agent_keystore_t *keystore, iot_agent_datastore_t *datastore, const char *hostname, uint32_t port, const pb_bytes_array_t *trusted_root_ca_certificates, const pb_bytes_array_t *client_certificate)`

Assemble a service descriptor for the connection to EdgeLock 2GO cloud service and write it to a datastore.

It is assumed that keys (either ECC or RSA) for authentication are stored in `keystore`.

If `client_certificate` is NULL, it is assumed that the client certificate is stored in `keystore` as well - in this case the resulting datastore contents will contain a reference to the certificate.

If `client_certificate` is not NULL, it shall point to a client certificate, which will be copied into the service descriptor.

Parameters

- **keystore** – **[in]** The input keystore
- **datastore** – **[out]** The datastore where to write
- **hostname** – **[in]** The hostname URL
- **port** – **[in]** The port number
- **trusted_root_ca_certificates** – **[in]** Buffer including the root certificates
- **client_certificate** – **[in]** Buffer including client certificate

Return values

- **IOT_AGENT_SUCCESS** – upon success
- **IOT_AGENT_FAILURE** – upon failure

`iot_agent_status_t iot_agent_utils_write_key_ref_pem(iot_agent_keystore_key_ref_t *gen_key_ref, const char *filename)`

Writes the key reference to PEM. The function is used when OpenSSL crypto is selected to write the key reference generated by the Open SSL to a file. The key reference provided as input depends on the used OpenSSL version.

- Version < 3.x.x: EVP_PKEY
- Version >= 3.x.x: OSSL_PARAM

Parameters

- **gen_key_ref** – **[in]** The key reference to be written
- **filename** – **[in]** name of file

Return values

- **IOT_AGENT_SUCCESS** – upon success
- **IOT_AGENT_FAILURE** – upon failure

`iot_agent_status_t iot_agent_utils_write_key_ref_pem_from_keystore(iot_agent_keystore_t *keyStore, const uint32_t keyid, const char *filename)`

Create a PEM file containing a key.

Parameters

- **keyStore** – [in] Key store context
- **keyid** – [in] Key id
- **filename** – [in] Name of the PEM file to be created

Return values

- **IOT_AGENT_SUCCESS** – upon success
- **IOT_AGENT_ERROR_CRYPTO_ENGINE_FAILED** – upon crypto operation failure
- **IOT_AGENT_ERROR_FILE_SYSTEM** – upon failure while creating or writing certificate file
- **IOT_AGENT_FAILURE** – upon failure

```
iot_agent_status_t iot_agent_utils_write_key_ref_service_pem(const iot_agent_context_t *ctx, const char *filename)
```

Create a PEM file containing a key reference to the key of the currently selected service.

Parameters

- **ctx** – [in] Context for the iot_agent
- **filename** – [in] Name of the PEM file to be created

Return values

- **IOT_AGENT_SUCCESS** – upon success
- **IOT_AGENT_FAILURE** – upon failure

Pre

A service needs to be selected before calling this function.

```
iot_agent_status_t nxp_iot_agent_unpad_iso7816d4(uint8_t *data, size_t *data_size)
```

Updates the length in order to remove padded data.

Parameters

- **data** – [inout] Buffer to be unpad
- **data_size** – [inout] The length of the buffer

Return values

- **IOT_AGENT_SUCCESS** – upon success
- **IOT_AGENT_FAILURE** – upon failure

6.8 EdgeLock 2GO Agent Examples

The usage of EdgeLock 2GO agent is demonstrated with the following example demo source code. The source code example is taken from the following file <SE05X_root_folder>/simw-top/nxp_iot_agent/ex/src/iot_agent_demo.c.

6.8.1 Initializaton of context and updating device configuration

Initialization of contexts of EdgeLock 2GO agent its keystores and datastores

```
agent_status = iot_agent_init(&iot_agent_context);
AGENT_SUCCESS_OR_EXIT();

/***************** Register keystore******/
```

```
agent_status = iot_agent_keystore_init(&keystore, EDGELOCK2GO_KEYSTORE_ID,
&iot_agent_platform_context);
AGENT_SUCCESS_OR_EXIT();

agent_status = iot_agent_register_keystore(&iot_agent_context, &keystore);
AGENT_SUCCESS_OR_EXIT();

***** Register datastore*****

// This is the datastore that holds connection information on how to connect to
// the EdgeLock 2GO cloud service itself. Typically this would be a persistent
// datastore which supports transactions to allow for atomic updates of
// the URL/port/etc. If a filesystem is available, here, we use a file-based
// datastore.
// Note, that the ID of the datastore is a given.
agent_status = iot_agent_datastore_init(&edgeLock2go_datastore,
nxp_iot_DatastoreIdentifiers_DATASTORE_EDGELOCK2GO_ID,
gszEdgeLock2GoDatastoreFilename,
&iot_agent_service_is_configuration_data_valid);
AGENT_SUCCESS_OR_EXIT();

// Configure the datastore using the EdgeLock 2GO hostname and port from one of
// the inputs:
// - Command line arguments
// - Environment variables
// - Existing valid datastore file
// - Macro defines
agent_status = iot_agent_utils_configure_edgeLock2go_datastore(&keystore,
&edgeLock2go_datastore, argc, argv);
AGENT_SUCCESS_OR_EXIT();

// For connecting to the EdgeLock 2GO cloud service, we also need to register
// the
// datastore that contains the information on how to connect there.
agent_status = iot_agent_set_edgeLock2go_datastore(&iot_agent_context,
&edgeLock2go_datastore);
AGENT_SUCCESS_OR_EXIT();

agent_status = iot_agent_datastore_init(&datastore, 0U, gszDatastoreFilename,
&iot_agent_service_is_configuration_data_valid);
AGENT_SUCCESS_OR_EXIT();

agent_status = iot_agent_register_datastore(&iot_agent_context, &datastore);
AGENT_SUCCESS_OR_EXIT();
```

Update device configuration:

```
agent_status = iot_agent_update_device_configuration(&iot_agent_context,
&status_report);

// We have a status report which can have some details on the operations that
// happened.
// This gives the opportunity to programmatically react on errors/failures. For
// demonstration
// purpose, print that information to the console here.
if ((agent_status == IOT_AGENT_SUCCESS || agent_status ==
IOT_AGENT_UPDATE_FAILED) && status_report.has_status) {
    iot_agent_print_status_report(&status_report);
}
```

```
AGENT_SUCCESS_OR_EXIT();
```

Iterate over configured services and access credentials and service configuration data:

```
if (!iot_agent_is_service_configuration_data_valid(&iot_agent_context))
{
    EXIT_STATUS_MSG(IOT_AGENT_FAILURE, "Not all configuration data is
valid");
}

// get total number of services
number_of_services = iot_agent_get_number_of_services(&iot_agent_context);
IOT_AGENT_INFO("Found configuration data for %d services.", (int)
number_of_services);
```

6.8.2 MQTT connection tests with Cloud Onboarding Provisioning

The example shows how to trigger the MQTT connection tests to the services when they are provisioned through the Cloud Onboarding Provisioning; the connection is triggered from following file: <SE05X_root_folder>/simw-top/nxp_iot_agent/ex/src/iot_agent_demo.c

```
#if (AX_EMBEDDED && defined(USE_RTO) && USE_RTO == 1) ||
(NXP_IOT_AGENT_HAVE_HOSTCRYPTO_OPENSSL)
    agent_status = iot_agent_verify_mqtt_connection(&iot_agent_context);
    AGENT_SUCCESS_OR_EXIT();
#endif
#endif //NXP_IOT_AGENT_HAVE_SSS
```

In case of FreeRTOS platform as LPC55S or FRDM_K64F the MQTT FreeRTOS client will be used for connection; implementation details can be found under <SE05X_root_folder>/simw-top/nxp_iot_agent/ex/src/utils/iot_agent_mqtt_freertos.c:

```
iot_agent_status_t
iot_agent_verify_mqtt_connection_for_service(iot_agent_context_t*
iot_agent_context,
    const nxp_iot_ServiceDescriptor* service_descriptor)
{
    iot_agent_status_t agent_status = IOT_AGENT_SUCCESS;
    agent_status = pubSub(iot_agent_context, service_descriptor);
    AGENT_SUCCESS_OR_EXIT_MSG("MQTT connection test failed");
exit:
    return agent_status;
}

iot_agent_status_t iot_agent_verify_mqtt_connection(iot_agent_context_t*
iot_agent_context)
{
    iot_agent_status_t agent_status = IOT_AGENT_SUCCESS;
    size_t number_of_services = 0U;
    nxp_iot_ServiceDescriptor service_descriptor =
    nxp_iot_ServiceDescriptor_init_default;

    number_of_services = iot_agent_get_number_of_services(iot_agent_context);
    for (size_t i = 0U; i < number_of_services; i++)
    {
        agent_status = iot_agent_select_service_by_index(iot_agent_context, i,
&service_descriptor);
```

```
AGENT_SUCCESS_OR_EXIT();

agent_status = iot_agent_verify_mqtt_connection_for_service(iot_agent_context,
&service_descriptor);
AGENT_SUCCESS_OR_EXIT();
}
exit:
iot_agent_free_service_descriptor(&service_descriptor);
return agent_status;
}
```

In case of Open SSL platform as iMX6 or iMX8 the MQTT Paho client will be used for connection; implementation details can be found under <SE05X_root_folder>/simw-top/nxp_iot_agent/ex/src/utils/iot_agent_mqtt_paho.c:

```
iot_agent_status_t
iot_agent_verify_mqtt_connection_for_service(iot_agent_context_t*
iot_agent_context, const nxp_iot_ServiceDescriptor* service_descriptor)
{
    iot_agent_status_t agent_status = IOT_AGENT_SUCCESS;
#ifndef OPENSSL_VERSION_NUMBER < 0x30000000
    int mkdir_check = 0;

#if IOT_AGENT_MQTT_CONNECTION_TEST_ENABLE
    mqtt_connection_params_t connection_params = { 0 };
    iot_agent_status_t mqtt_status = IOT_AGENT_SUCCESS;
#endif
    //write to files

    char config_filename[256];
    char certificate_filename[256];
    char keyref_filename[256];
    char server_cert_filename[256];
    char metadata_filename[256];

    const char* service_type_str;
    char service_dir[64];

    uint32_t keystore_id = service_descriptor->client_key_sss_ref.endpoint_id;
    iot_agent_keystore_t* keystore = NULL;
    agent_status = iot_agent_get_keystore_by_id(iot_agent_context, keystore_id,
    &keystore);
    AGENT_SUCCESS_OR_EXIT();

    agent_status = iot_agent_keystore_open_session(keystore);
    AGENT_SUCCESS_OR_EXIT();

    agent_status = iot_agent_service_get_service_type_as_string(service_descriptor,
    &service_type_str);
    AGENT_SUCCESS_OR_EXIT();

    if (ACCESS(goutput_directory, R_OK) != 0)
    {
        mkdir_check = DO_MKDIR(goutput_directory);
        ASSERT_OR_EXIT(mkdir_check == 0);
    }
}
```

```
ASSERT_OR_EXIT_MSG(snprintf(service_dir, sizeof(service_dir), "%s%c%s%c",
goutput_directory, path_separator, service_type_str, path_separator) >= 0,
"Error in creating output folder string");
if (ACCESS(service_dir, R_OK) != 0)
{
    mkdir_check = DO_MKDIR(service_dir);
    ASSERT_OR_EXIT(mkdir_check == 0);
}

ASSERT_OR_EXIT_MSG(snprintf(config_filename, sizeof(config_filename), "%s"
SERVICE_CONFIGURATION_PATTERN, service_dir, service_descriptor->identifier) >=
0, "Error in creating filename string");
ASSERT_OR_EXIT_MSG(snprintf(metadata_filename, sizeof(metadata_filename), "%s"
SERVICE_METADATA_PATTERN, service_dir, service_descriptor->identifier) >= 0,
"Error in creating filename string");
ASSERT_OR_EXIT_MSG(snprintf(keyref_filename, sizeof(keyref_filename), "%s"
SERVICE_KEYREF_PATTERN, service_dir, service_descriptor->identifier) >= 0,
"Error in creating filename string");
ASSERT_OR_EXIT_MSG(snprintf(certificate_filename,
sizeof(certificate_filename), "%s" SERVICE_CERTIFICATE_PATTERN, service_dir,
service_descriptor->identifier) >= 0, "Error in creating filename string");
ASSERT_OR_EXIT_MSG(snprintf(server_cert_filename,
sizeof(server_cert_filename), "%s" SERVICE_SERVER_CERT_PATTERN, service_dir,
service_type_str, service_descriptor->identifier) >= 0, "Error in creating
filename string");

//create service files
char relative_certificate_filename[128];
char relative_keyref_filename[128];
char relative_server_cert_filename[128];
ASSERT_OR_EXIT_MSG(snprintf(relative_certificate_filename,
sizeof(relative_certificate_filename), SERVICE_CERTIFICATE_PATTERN,
service_descriptor->identifier) >= 0, "Error in creating filename string");
ASSERT_OR_EXIT_MSG(snprintf(relative_keyref_filename,
sizeof(relative_keyref_filename), SERVICE_KEYREF_PATTERN, service_descriptor-
>identifier) >= 0, "Error in creating filename string");
ASSERT_OR_EXIT_MSG(snprintf(relative_server_cert_filename,
sizeof(relative_server_cert_filename), SERVICE_SERVER_CERT_PATTERN,
service_type_str, service_descriptor->identifier) >= 0, "Error in creating
filename string");

if (IOT_AGENT_SUCCESS != write_service_configuration(service_descriptor,
config_filename,
relative_certificate_filename, relative_keyref_filename,
relative_server_cert_filename))
{
    write_error_logs(config_filename);
}
else
{
    LOG_D("Created service configuration file [%s]", config_filename);
}

if (IOT_AGENT_SUCCESS != write_service_metadata(service_descriptor,
metadata_filename))
{
    write_error_logs(metadata_filename);
}
else
```

```
{  
    LOG_D("Created metadata file [%s]", metadata_filename);  
  
    if (service_descriptor->server_certificate != NULL) {  
        if (IOT_AGENT_SUCCESS !=  
            iot_agent_utils_write_certificate_pem(service_descriptor->server_certificate-  
>bytes,  
            (size_t)service_descriptor->server_certificate->size, server_cert_filename))  
        {  
            write_error_logs(server_cert_filename);  
        }  
    }  
    else  
    {  
        LOG_D("Created server root certificate file [%s]", server_cert_filename);  
    }  
}  
  
if (service_descriptor->client_certificate != NULL) {  
    if (IOT_AGENT_SUCCESS !=  
        iot_agent_utils_write_certificate_pem(service_descriptor->client_certificate-  
>bytes,  
        (size_t)service_descriptor->client_certificate->size, certificate_filename))  
    {  
        write_error_logs(certificate_filename);  
    }  
} else  
{  
    LOG_D("Created service client certificate file [%s]", certificate_filename);  
}  
}  
  
if (IOT_AGENT_SUCCESS !=  
iot_agent_utils_write_key_ref_service_pem(iot_agent_context, keyref_filename))  
{  
    write_error_logs(keyref_filename);  
}  
else  
{  
    LOG_D("Created service keyref file [%s]", keyref_filename);  
}  
//write to files - end  
  
#if IOT_AGENT_MQTT_CONNECTION_TEST_ENABLE  
iot_agent_keystore_close_session(keystore);  
  
int network_status = network_openssl_engine_session_connect_mqtt();  
ASSERT_OR_EXIT_MSG(network_status == NETWORK_STATUS_OK,  
"network_openssl_engine_session_connect failed with 0x%08x.", network_status);  
  
connection_params.keypath = keyref_filename;  
connection_params.devcert = certificate_filename;  
connection_params.rootpath = server_cert_filename;  
mqtt_status = iot_agent_mqtt_test(service_descriptor, &connection_params);  
  
network_status = network_openssl_engine_session_disconnect_mqtt();  
ASSERT_OR_EXIT_MSG(network_status == NETWORK_STATUS_OK,  
"network_openssl_engine_session_connect failed with 0x%08x.", network_status);  
  
agent_status = iot_agent_keystore_open_session(keystore);
```

```

AGENT_SUCCESS_OR_EXIT();
#endif
exit:
    if (mqtt_status != IOT_AGENT_SUCCESS) {
        return mqtt_status;
    }
#endif //if (OPENSSL_VERSION_NUMBER < 0x30000000)
    return agent_status;
}

iot_agent_status_t iot_agent_verify_mqtt_connection(iot_agent_context_t*
    iot_agent_context)
{
    iot_agent_status_t agent_status = IOT_AGENT_SUCCESS;
    size_t number_of_services = 0U;
    nxp_iot_ServiceDescriptor service_descriptor =
    nxp_iot_ServiceDescriptor_init_default;

    number_of_services = iot_agent_get_number_of_services(iot_agent_context);
    delete_old_service_files_folders(goutput_directory);
    AGENT_SUCCESS_OR_EXIT();

    for (size_t i = 0U; i < number_of_services; i++)
    {
        agent_status = iot_agent_select_service_by_index(iot_agent_context, i,
&service_descriptor);
        AGENT_SUCCESS_OR_EXIT();

        agent_status = iot_agent_verify_mqtt_connection_for_service(iot_agent_context,
&service_descriptor);
        AGENT_SUCCESS_OR_EXIT();
    }
exit:
    iot_agent_free_service_descriptor(&service_descriptor);
    return agent_status;
}

```

6.8.3 MQTT connection tests with Remote Trust Provisioning

The example shows how to trigger the MQTT connection tests to the services when they are provisioned through the Remote Trust Provisioning; the connection is triggered from following file: <SE05X_root_folder>/simw-top/nxp_iot_agent/ex/src/iot_agent_demo.c

```

#if ((AX_EMBEDDED && defined(USE_RTOS) && USE_RTOS == 1) ||
    (NXP_IOT_AGENT_HAVE_HOSTCRYPTO_OPENSSL)) && (IOT_AGENT_MQTT_ENABLE == 1)
// example code for MQTT conneciton when provisioning services as RTP objects
iot_agent_cleanup_mqtt_config_files_cos_over_rtp();
AGENT_SUCCESS_OR_EXIT();
#endif AWS_ENABLE
// modify the connection parameters to allow connection to your AWS account
agent_status =
iot_agent_get_mqtt_service_descriptor_for_aws(&iot_agent_context,
&aws_service_descriptor);
AGENT_SUCCESS_OR_EXIT();
// in case of AWS device auto-registration the MQTT connection will fail on the
first connection; in this

```

```

// use case add additional call to the funcion
iot_agent_verify_mqtt_connection_cos_over_rtp
agent_status =
iot_agent_verify_mqtt_connection_cos_over_rtp(&iot_agent_context,
&aws_service_descriptor);
AGENT_SUCCESS_OR_EXIT();
#endif
#if AZURE_ENABLE
agent_status = iot_agent_get_service_descriptor_for_azure(&iot_agent_context,
&azure_service_descriptor);
AGENT_SUCCESS_OR_EXIT();
agent_status =
iot_agent_verify_mqtt_connection_cos_over_rtp(&iot_agent_context,
&azure_service_descriptor);
AGENT_SUCCESS_OR_EXIT();
#endif
#endif

```

To enable the MQTT connection for the RTP use case some manual adjustments needs to be done in the Agent code; the use case can be enabled by setting the variable IOT_AGENT_MQTT_ENABLE to 1 in the file <SE05X_root_folder>/simw-top/nxp_iot_agent/inc/nxp_iot_agent_config.h

The user has to setup some configuration parameters for the service to which he want to connect; this can be done by changing some definitions in the file: <SE05X_root_folder>/simw-top/nxp_iot_agent/ex/inc/iot_agent_demo_config.h

```

// CHANGE THIS : Set to 1 or 0 to enable or disable AWS MQTT connection
#define AWS_ENABLE 1U

// CHANGE THIS : fill with key pair and device certificate object IDs as defined
// on EL2GO when generating them
#ifndef AWS_SERVICE_KEY_PAIR_ID
#define AWS_SERVICE_KEY_PAIR_ID      0x83000101U
#endif

#ifndef AWS_SERVICE_DEVICE_CERT_ID
#define AWS_SERVICE_DEVICE_CERT_ID 0x83000102U
#endif

// CHANGE THIS: the AWS hostname to which the device will connect
#define AWS_HOSTNAME "aw9969rp3sm22-ats.iot.eu-central-1.amazonaws.com"

// optional: the client ID will be by default set dynamically to the Common Name
// of the
// device leaf certificate; is possible to hardocode it, by uncommenting the
// below line and assign
// the desired value
// #define AWS_CLIENT_ID "awsrtptest-000000000001e6e-0000"

// CHANGE THIS: set the desired service ID; is used internally by the MQTT
// client, should be unique
// for every service
#define AWS_SERVICE_ID 101

// CHANGE THIS : Set to 1 or 0 to enable or disable Azure MQTT connection
#define AZURE_ENABLE 1U

// CHANGE THIS : fill with key pair and device certificate object IDs as defined
// on EL2GO when generating them

```

```
#ifndef AZURE_SERVICE_KEY_PAIR_ID
#define AZURE_SERVICE_KEY_PAIR_ID      0x83000211U
#endif

#ifndef AZURE_SERVICE_DEVICE_CERT_ID
#define AZURE_SERVICE_DEVICE_CERT_ID   0x83000212U
#endif

// CHANGE THIS : set the ID scope and the global endpoint as defined in the
// Azure DPS account
#define AZURE_ID_SCOPE                "0ne004510C6"
#define AZURE_GLOBAL_DEVICE_ENDPOINT  "global.azure-devices-provisioning.net"

// optional: the registration ID will be by default set dynamically to the
// Common Name of the
// device leaf certificate; is possible to hardocode it, by uncommenting the
// below line and assign
// the desired value
// #define AZURE_REGISTRATION_ID "azurerptptest-000000000001e51-0000"

// CHANGE THIS: set the desired service ID; is used internally by the MQTT
// client, should be unique
// for every service
#define AZURE_SERVICE_ID 102
```

Depending on the service type the Service Descriptor should be set to include the correct connection data and object IDs of the Key Pair and X.509 device leaf certificate associated with the service. The following functions should be modified in the file <SE05X_root_folder>/simw-top/nxp_iot_agent/ex/src/iot_agent_demo.c

```
iot_agent_status_t
iot_agent_get_mqtt_service_descriptor_for_aws(iot_agent_context_t*
iot_agent_context,
nxp_iot_ServiceDescriptor* service_descriptor)
{
    iot_agent_status_t agent_status = IOT_AGENT_SUCCESS;

    ASSERT_OR_EXIT_MSG(service_descriptor != NULL, "Service descriptor is null");

    // AWS Service type
    service_descriptor->identifier = AWS_SERVICE_ID;
    service_descriptor->has_service_type = true;
    service_descriptor->service_type = nxp_iot_ServiceType_AWSSERVICE;

    // service key pair
    service_descriptor->has_client_key_sss_ref = true;
    service_descriptor->client_key_sss_ref.object_id = AWS_SERVICE_KEY_PAIR_ID;
    service_descriptor->client_key_sss_ref.has_type = false;
    service_descriptor->client_key_sss_ref.type = nxp_iot_EndpointType_INVALID;
    service_descriptor->client_key_sss_ref.has_endpoint_id = true;
    service_descriptor->client_key_sss_ref.endpoint_id = EDGELOCK2GO_KEYSTORE_ID;
    service_descriptor->client_key_sss_ref.has_object_id = true;

    // client certificate
    service_descriptor->has_client_certificate_sss_ref = true;
    service_descriptor->client_certificate_sss_ref.object_id =
AWS_SERVICE_DEVICE_CERT_ID;
    service_descriptor->client_certificate_sss_ref.has_type = false;
```

```
service_descriptor->client_certificate_sss_ref.type =
nxp_iot_EndpointType_INVALID;
service_descriptor->client_certificate_sss_ref.has_endpoint_id = true;
service_descriptor->client_certificate_sss_ref.endpoint_id =
EDGELOCK2GO_KEYSTORE_ID;
service_descriptor->client_certificate_sss_ref.has_object_id = true;

// AWS server certificate
service_descriptor->server_certificate =
(pb_bytes_array_t*)&aws_root_server_cert;
service_descriptor->has_server_certificate_sss_ref = false;

// AWS MQTT connection parameters
char hostname[] = AWS_HOSTNAME;
iot_agent_fill_service_char_array(&service_descriptor->hostname, hostname,
sizeof(hostname));

service_descriptor->has_port = true;
service_descriptor->port = 8883;
service_descriptor->has_timeout_ms = true;
service_descriptor->timeout_ms = 0x4E20;
service_descriptor->has_protocol = true;
service_descriptor->protocol = nxp_iot_ServiceProtocolType_MQTT;

#ifdef AWS_CLIENT_ID
char client_id[] = AWS_CLIENT_ID;
iot_agent_fill_service_char_array(&service_descriptor->client_id, client_id,
sizeof(client_id));
#else
service_descriptor->client_id = malloc(COMMON_NAME_MAX_SIZE);
memset(service_descriptor->client_id, 0, COMMON_NAME_MAX_SIZE);
agent_status = iot_agent_utils_get_certificate_common_name(iot_agent_context,
service_descriptor, service_descriptor->client_id, COMMON_NAME_MAX_SIZE);
AGENT_SUCCESS_OR_EXIT();
#endif

service_descriptor->username = NULL;
service_descriptor->password = NULL;

// fill metadata if any
service_descriptor->customer_metadata_json = NULL;

exit:
return agent_status;
}

iot_agent_status_t
iot_agent_get_service_descriptor_for_azure(iot_agent_context_t*
iot_agent_context,
nxp_iot_ServiceDescriptor* service_descriptor)
{
    iot_agent_status_t agent_status = IOT_AGENT_SUCCESS;

    ASSERT_OR_EXIT_MSG(service_descriptor != NULL, "Service descriptor is null");

// Azure Service type
service_descriptor->identifier = AZURE_SERVICE_ID;
service_descriptor->has_service_type = true;
service_descriptor->service_type = nxp_iot_ServiceType_AZURESERVICE;
```

```
// service key pair
service_descriptor->has_client_key_sss_ref = true;
service_descriptor->client_key_sss_ref.object_id = AZURE_SERVICE_KEY_PAIR_ID;
service_descriptor->client_key_sss_ref.has_type = false;
service_descriptor->client_key_sss_ref.type = nxp_iot_EndpointType_INVALID;
service_descriptor->client_key_sss_ref.has_endpoint_id = true;
service_descriptor->client_key_sss_ref.endpoint_id = EDGELOCK2GO_KEYSTORE_ID;
service_descriptor->client_key_sss_ref.has_object_id = true;

// service client certificate
service_descriptor->has_client_certificate_sss_ref = true;
service_descriptor->client_certificate_sss_ref.object_id =
AZURE_SERVICE_DEVICE_CERT_ID;
service_descriptor->client_certificate_sss_ref.has_type = false;
service_descriptor->client_certificate_sss_ref.type =
nxp_iot_EndpointType_INVALID;
service_descriptor->client_certificate_sss_ref.has_endpoint_id = true;
service_descriptor->client_certificate_sss_ref.endpoint_id =
EDGELOCK2GO_KEYSTORE_ID;
service_descriptor->client_certificate_sss_ref.has_object_id = true;

// Azure server certificate
service_descriptor->server_certificate =
(pb_bytes_array_t*)&azure_root_server_cert;
service_descriptor->has_server_certificate_sss_ref = false;

// Azure MQTT connection parameters
char azure_id_scope[] = AZURE_ID_SCOPE;
iot_agent_fill_service_char_array(&service_descriptor->azure_id_scope,
azure_id_scope, sizeof(azure_id_scope));

// the registration ID will be by default set dynamically to the Common Name of
// the device leaf certificate;
// it is possible to hardcode it, by uncommenting the AZURE_REGISTRATION_ID in
// the configuration file
#ifdef AZURE_REGISTRATION_ID
char azure_registration_id[] = AZURE_REGISTRATION_ID;
iot_agent_fill_service_char_array(&service_descriptor->azure_registration_id,
azure_registration_id, sizeof(azure_registration_id));
#else
service_descriptor->azure_registration_id = malloc(COMMON_NAME_MAX_SIZE);
memset(service_descriptor->azure_registration_id, 0, COMMON_NAME_MAX_SIZE);
agent_status = iot_agent_utils_get_certificate_common_name(iot_agent_context,
service_descriptor, service_descriptor->azure_registration_id,
COMMON_NAME_MAX_SIZE);
AGENT_SUCCESS_OR_EXIT();
#endif

char azure_global_device_endpoint[] = AZURE_GLOBAL_DEVICE_ENDPOINT;
iot_agent_fill_service_char_array(&service_descriptor-
>azure_global_device_endpoint, azure_global_device_endpoint,
sizeof(azure_global_device_endpoint));

service_descriptor->has_port = false;
service_descriptor->port = 0;
service_descriptor->has_timeout_ms = true;
service_descriptor->timeout_ms = 0x4E20;
service_descriptor->has_protocol = false;
service_descriptor->username = NULL;
service_descriptor->password = NULL;
```

```
// fill metadata if any
service_descriptor->customer_metadata_json = NULL;

exit:
    return agent_status;
}
```

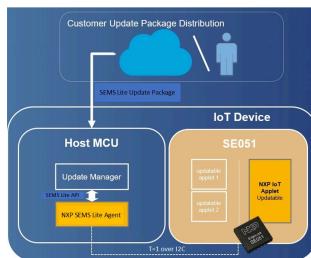
In the case of AWS auto-registration the Intermediate certificate used to sign the device leaf certificate needs to be presented to the cloud when doing the registration; EdgeLock 2GO offers the possibility to provision the intermediate CA together with the leaf certificate in one combined object. The MQTT Client on Embedded Linux devices is able to automatically load the combined object and execute the auto-registration. On FreeRTOS devices the dynamic loading is not implemented and the intermediate certificate must be provided in the code; to achieve this, the value of the #define keyJITR_DEVICE_CERTIFICATE_AUTHORITY_PEM in file <SE05X_root_folder>/simw-top/demos/ksdk/common/aws_clientcredential_keys.h must be filled with the value of the intermediate certificate in PEM format.

7 SEMS Lite Agent

7.1 SEMS Lite Overview (Only for SE051)

SEMSLite enables the deployment and update of applets on SE in the field. When the applets are being upgraded, the data of previous/new applet remains inside the secure element only. Thus it securely preserves the device applet data.

Note: SEMS Lite is only supported for SE051.



7.1.1 Update Manager (from customer)

The role of update manager is as under:

- Securely downloading *update package* from its back-end
- Defining right time for update (depending on context: battery status, IoT device usage profile, etc.)
- Managing switching between current and new applet
- Optionally reporting update status to back-end

We provide some examples for update manager ([Section 5.1.1.11SEMS Demos](#))

7.1.2 SEMS Lite Agent (from NXP)

The role of SEMS Lite Agent is an under:

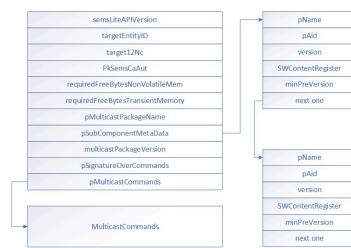
- Provide functional APIs to Update Manager
- It is a library/module that helps the *Update Manager* to query/know the state of the system

- Use the *update package* as received by the *Update Manager*, and update the Applet
- Track the update progress / interrupted updates
- Optionally retrieve loading receipt from the secure element.

7.2 Update Package

SEMS Lite agent gets an *update package* from the *update manager*. The upgrade package contains package meta data header and set of APDUs(MulticastCommands) to be sent to the SE for update. It may optionally also contain host control commands (e.g. Reset the secure element).

Format of the *update package*:



NXP provides customer package meta data and Multicast commands in JSON file. NXP also provides generator tool to turn the JSON file to binary file and .c/.h files. Customer can decide which one to be used by *update manager* according to their platform. *Update manager* should create *update package* from these files. This chapter has provided examples for the use of generator. It also provides links to the demos which will use the generated files to create *update package*.

7.2.1 Package Meta Data

Package meta data includes information such as version and memory requirement. These information will help SEMS Lite agent to preprocess package before loading APDU commands to SE.

- MulticastPackageFormatVersion: Version information of this json Format for MulticastPackages.
- TargetEntityID: Entity ID, 16bytes long Binary Coded Decimal, of the target device where this MulticastPackage is intended to be executed on. It is an identifier of the key-set of the Multicast Applet Loader.
- Target12nc: Target 12nc is a 12 digit numerical code identifying the target device where this MulticastPackage is intended to be executed on, as known to customers and used on EdgeLock2Go to identify device types.
- PkSemsCaAut: The PkSemsCaAut key found in the SEMS_SP_AUT.txt file as PK.AP.AUT.
- requiredFreeBytesNonVolatileMemory: Minimum required free Non Volatile memory in bytes that have to be available on the target device before execution of this MulticastPackage.
- requiredFreeBytesTransientMemory: Minimum required free transient (RAM) memory in bytes that have to be available on the target device before execution of this MulticastPackage.
- MulticastPackageName: Giving a descriptive name to the complete Multicast Package.
- MulticastPackageVersion: Version information of this MulticastPackage, describing the sum version over all contained content.
- SubComponentMetaData: A list of subcomponents of this MulticastPackage, designating all Executable Load Files (ELFs) Contained. It usually contains one entry, but can have multiple in the case multiple dependent ELF's get modified. This list can be empty, e.g. for a KeyRotation or deletion of content.
- SubComponentMetaData name: A human readable name for this subcomponent.

- SubComponentMetaData aid: The Application Identifier (AID) of the Executable Load File (ELF) which makes up the content of this subcomponent. This is stored as string to have it formatted in upper-case hexadecimal and therefore recognizable form.
- SubComponentMetaData version: Version information of this subcomponent.
- SubComponentMetaData SWContentRegister: NXP Software Content Register, specifies the Outgoing License, mostly: LA_OPT_NXP_Software_License.
- SubComponentMetaData minimumPreviousVersion: Minimum version number of this subcomponent as installed on the secure element before this script is executed. If this field is omitted there is no minimum version requirement, e.g. initial Installation of an applet.
- SignatureOverCommands: The signature over the multicast commands in an machine readable form. So it does not have to be parsed from the script commands. String encoding (upper-case hexadecimal) is chosen here, as many json parsers can not handle such large integer values.
- MulticastCommands: The complete Multicast Applet Loader Script (certificate, signature, encrypted and signed commands) in ls-cgt format, encoded in base64.

7.2.2 MulticastCommands In Protocol Buffer Format

Note:

Advanced information

The information below is for advanced users and kept here for the sake of completeness of information. This section can be skipped.

Technically, the APDUS(MulticastCommands) in update package is encoded in Protocol Buffers format (<https://developers.google.com/protocol-buffers/>) as defined below

```
/* Full request from cloud/host to the agent */
message Requests {
    /* An array of RequestPayload */
    repeated RequestPayload payload = 1;
}

/* Consolidated response from the agent to the cloud/host */
message Responses {
    /* An array of ResponsePayload */
    repeated ResponsePayload responses = 1;
}

/* The request payload would be either of these */
message RequestPayload {
    oneof payload {
        /* Hello from EdgeLock 2GO cloud service to the EdgeLock 2GO agent.
         *
         * Used only by EdgeLock 2GO agent. */
        AgentHelloRequest hello = 1;
        /* Good Bye from EdgeLock 2GO cloud service to the EdgeLock 2GO agent.
         *
         * Used only by EdgeLock 2GO agent. */
        AgentGoodbyeRequest goodbye = 2;
        AgentCrlRequest crl = 3;
        /* An APDU to a SE.
         *
         * Used both by EdgeLock 2GO agent and SEMS Lite agent. */
        ApduRequest apdu = 20;
        /* Message to read and write configuration data.
         *
         * Used both by EdgeLock 2GO agent and SEMS Lite agent. */
        ConfigurationRequest config = 21;
    }
}
```

```
* Used only by EdgeLock 2GO agent. */
DatastoreRequest datastore = 30;
/* Message to control the host software on the device.
*/
/* Not implemented.*/
RpcRequest rpc = 40;
/*
*/
/* Used only by SEMS Lite Agent */
HostControlCmdRequest hostCmd = 50;%%CRLF
SP%/* Contains requests for a keystore implementing a PSA-style interface
*/
/* Used only by EdgeLock 2GO agent.*/
PsaRequest psa = 60;%%CRLF
SP%/* Contains requests for the Proxy hello message
*/
/* Used only by EdgeLock 2GO agent.*/
HelloProxyRequest helloProxy = 70;
}

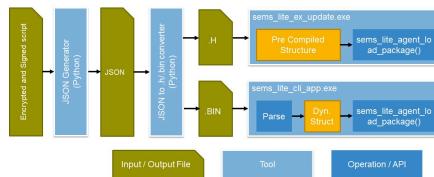
message ApduRequest {%%CRLF
SP%/* Array of bytes to be sent to the SE */
optional bytes message = 100;%%CRLF
SP%/* What response is expected from the SE for this APDU?
*/
/* If this field is skipped from protobuf, then,
 * only 0x9000 is expected from the SE
*/
/* There are potentially multiple distinct values
 * are expected from the SE, then this
 * can be an array of those values.
*/
/* e.g. expectation: [ 0x9000, 0x6A82 ] */
/* For complex schemes where a every big range/mask
 * is expected, e.g. 6AXX, 6DX then the upper
 * 16 bits are treated as mask.
*/
/* e.g. expectation: [ 0x00FF6A00, 0x00FF6D00 ] */
/* This field is unused by EdgeLock 2GO cloud service.
*/
repeated uint32 expectation = 101;
}%%CRLF
CRLF%/* Response from the SE to host */
message ApduResponse {
/* Byte array */
optional bytes message = 100;
}
```

7.2.3 Generator Tools

NXP provides 2 generator tools in `semslit/tools/sems-lite-generator` directory:

- JSON Generator: A tool to convert the output of CGT tool to a JSON format output

- SEMS Lite generator: A tool to convert JSON output to binary file and .c/.h files which will be used by *update manager*



7.2.3.1 JSON Generator

Usage: MulticastPackageCli.py [-h] –pkg_format [PKG_FORMAT_VERSION] –config_file [CONFIG_FILE] –script_file [SCRIPT_FILE] –out [OUT]

Arguments:

- pkg_format [PKG_FORMAT_VERSION]**

Package format version. It can be either 1.2 or 1.3.

- config_file [CONFIG_FILE]**

Config File for MulticastPackage generation. It should follow the format defined in `semsslit/tools/sems-lite-generator/schema/MulticastPackage.jsonschema`. NXP provides an example in `semsslit/tools/sems-lite-generator/config/ExampleConfig_x_x.json`

- script_file [SCRIPT_FILE]**

Encrypted and Signed script as output by the ls-cgt tool.

- out [OUT]**

Output MulticastPackage json file.

Example:

```
python ./MulticastPackageCli.py –pkg_format 1.3 –config_file .config.json –script_file .encrypted.txt –out .Upgrade_IoTApplet.json
```

7.2.3.2 SEMS Lite Generator

Usage: generate.py [-h] [-i INPUT_JSON] [-o OUTPUT_PATH] [-n NAME] [-p PROTOC_PATH]

Process sems-lite-generator arguments

Arguments:

-h --help	show this help message and exit
-i --input_json INPUT_JSON	input json file
-o --output_path OUTPUT_PATH	output folder path
-n	stem name of output files. By default, it would use the same name as input json file.

--name NAME	
-p	protoc file path. Use tools/mw_onverter/protoc.exe by default
--protoc_path PROTOC_PATH	

Example:

```
python ./generate.py -i ./Upgrade_NXP-IoTApplet-7.3.json -o ./
```

The generated files:

- Upgrade_NXP-IoTApplet-7.3.bin: Binary file. It can be used for platforms that have a file system (Windows, Linux, etc). It is encoded in TLV as following:

```

0x21 Len multicastPackage
|- 0x22 Len Major Minor MulticastPackageFormatVersion
|- 0x23 Len Binary TargetEntityID
|- 0x2f Len Binary Target12nc
|- 0x31 Len Binary PkSemsCaAut
|- 0x24 Len u32/u16 requiredFreeBytesNonVolatileMemory
|- 0x25 Len u32/u16 requiredFreeBytesTransientMemory
|- 0x26 Len String MulticastPackageName
|- 0x27 Len Major Minor MulticastPackageVersion
|- 0x28 Len SubComponentMetaData
  |- 0x2B Len String SubComponentMetaData1.name
  |- 0x2C Len Binary SubComponentMetaData1.aid
  |- 0x2D Len Major Minor SubComponentMetaData1.version
  |- 0x30 Len String SubComponentMetaData1.SWContentRegister
  |- 0x2E Len Major Minor
SubComponentMetaData1.minimumPreviousVersion
  |- 0x2B Len String SubComponentMetaData2.name
  |- 0x2C Len Binary SubComponentMetaData2.aid
  |- 0x2D Len Major Minor SubComponentMetaData2.version
  |- 0x30 Len String SubComponentMetaData2.SWContentRegister
  |- 0x2E Len Major Minor
SubComponentMetaData2.minimumPreviousVersion
...
  |- 0x2B Len String SubComponentMetaDataN.name
  |- 0x2C Len Binary SubComponentMetaDataN.aid
  |- 0x2D Len Major Minor SubComponentMetaDataN.version
  |- 0x30 Len String SubComponentMetaDataN.SWContentRegister
  |- 0x2E Len Major Minor
SubComponentMetaDataN.minimumPreviousVersion
  |- 0x29 Len Binary SignatureOverCommands
  |- 0x2A Len Binary MulticastCommands

```

- Upgrade_NXP-IoTApplet-7.3.c and .h: These 2 files instantiate *update package*. They can be integrated into customer tools and can be used for all platforms.

We provide examples for how to use the generated files: [Section 5.1.1.11SEMS Demos](#)

7.2.3.3 Generator Tool Version Compatibility

NXP provides both SEMS Lite generate tool and JSON format in semslite\tools folder. In SEMS Lite 3.0.0 (Refer to semslite\version_info.txt) the JSON package format version is 1.3. But the generate tool in SEMS Lite 3.0.0 can work with the JSON schema of both 1.2 and 1.3.

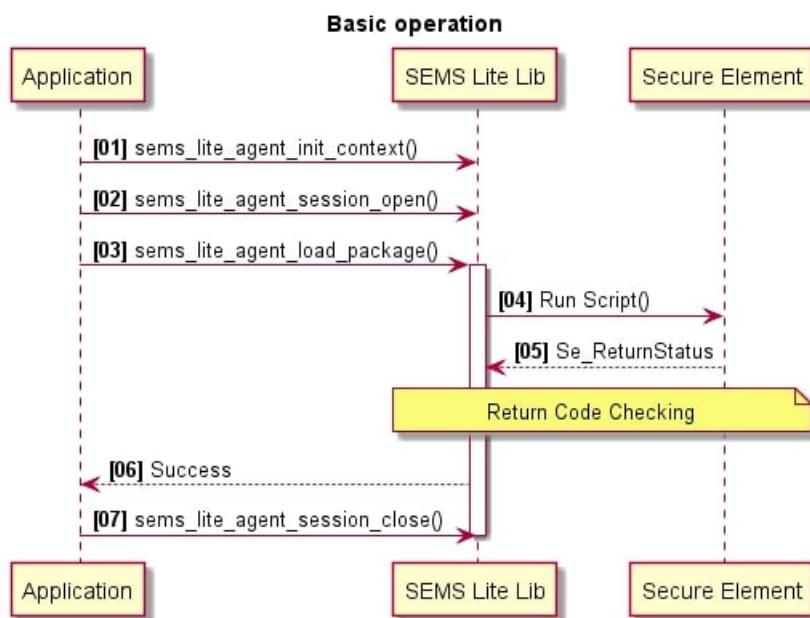
	JSON package format 1.1	JSON package format 1.2	JSON package format 1.3
SEMS Lite 1.0.0	Supported	Not supported	Not supported
SEMS Lite 2.0.0	Not supported	Supported	Not supported
SEMS Lite 3.0.0	Not supported	Supported	Supported

7.3 SEMS Lite Agent Usage

7.3.1 Case 1: Basic Usage

Basic API usage documentation for using SEMS Lite agent is as under.

1. Call `sems_lite_agent_init_context()` to do context initialization.
2. Call `sems_lite_agent_session_open()` to open channel to SE and select SEMS Lite applet.
3. Call `sems_lite_agent_load_package()` to load upgrade script.
4. SEMS Lite agent would internally query the tear status from SE051, if there was a power interrupt during upgradation(tearing event).
5. SEMS Lite agent load the script from package to SE051 and get response from SE051.
6. SEMS Lite agent checking the response and return "Success" to user.
7. User call `sems_lite_agent_session_close()` to close channel.

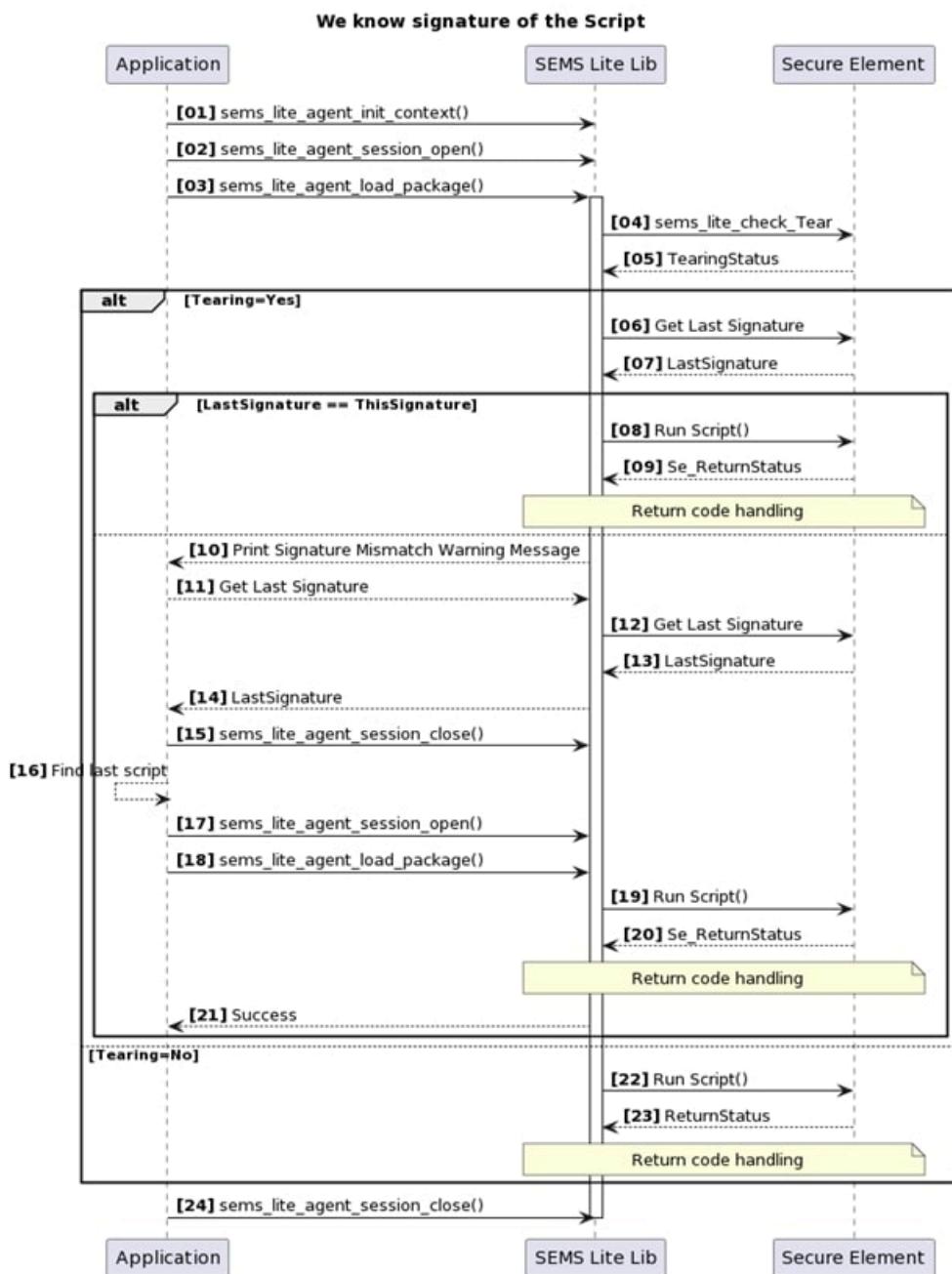


7.3.2 Case 2: In case there was a tearing

In case of tear which is due to system power off during uploading, user would receive return code "DoRerun" when he try to load new package. Use must load the torn again.

1. Call `sems_lite_agent_init_context()`, `sems_lite_agent_session_open()` and `sems_lite_agent_load_package()` as those in [Case 1: Basic Usage](#).

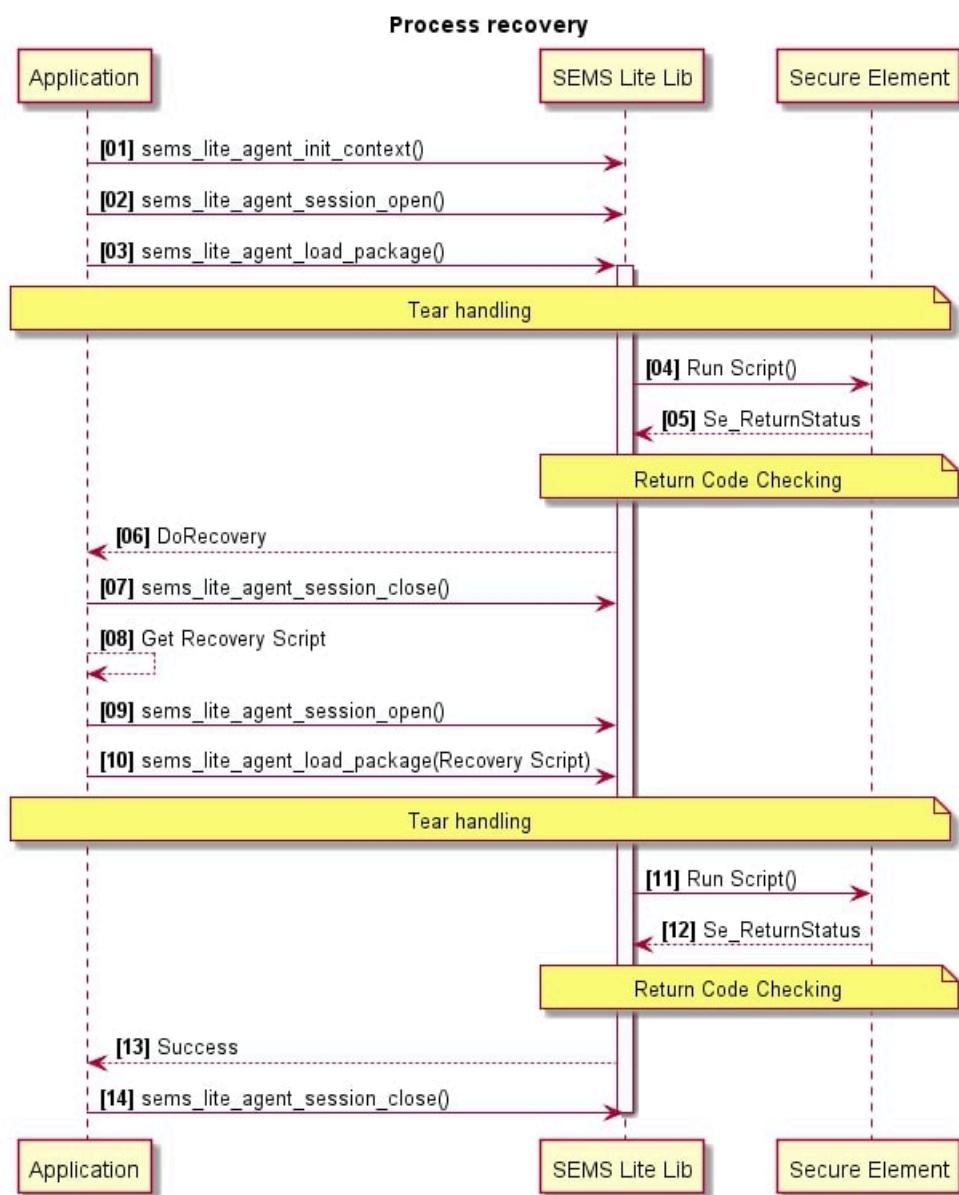
2. SEMS Lite agent would query tear status from SE.
3. In case tear happens, SEMS Lite agent would get signature of tear script from SE and compare it to the signature of loaded script.
4. If these 2 signatures are same, it means the loaded script is the re-run script. SEMS Lite agent load it to SE.
5. If the signatures are different, SEMS Lite agent would inform user by printing a warning message "SEMS Lite signature not match!!!". It will also output "Current package signature" and "Expected signature" which can be used in next step.
6. User must find the correct script by signature and load it again if signature mismatch warning message is printed.



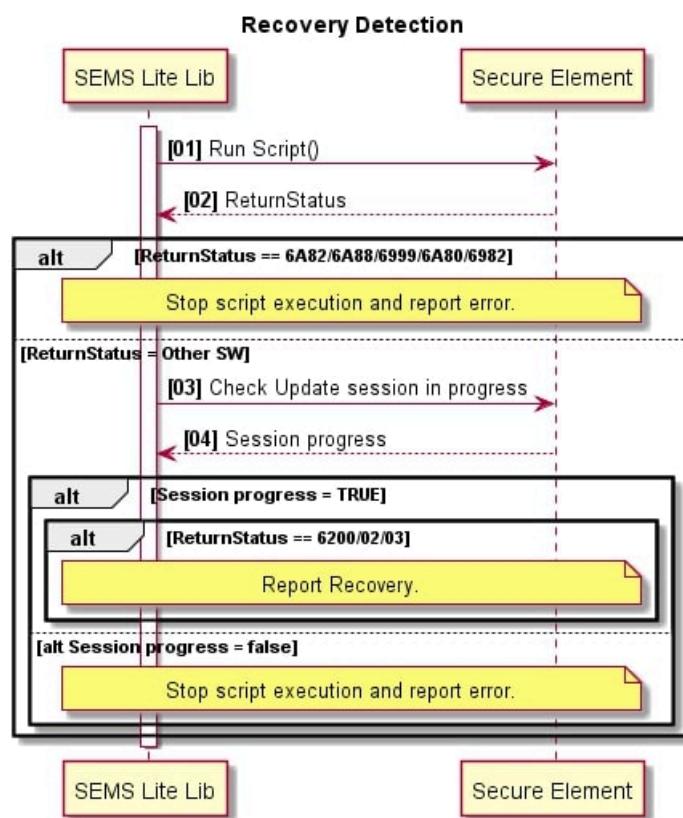
7.3.3 Case 3: Doing a recovery after a failed update

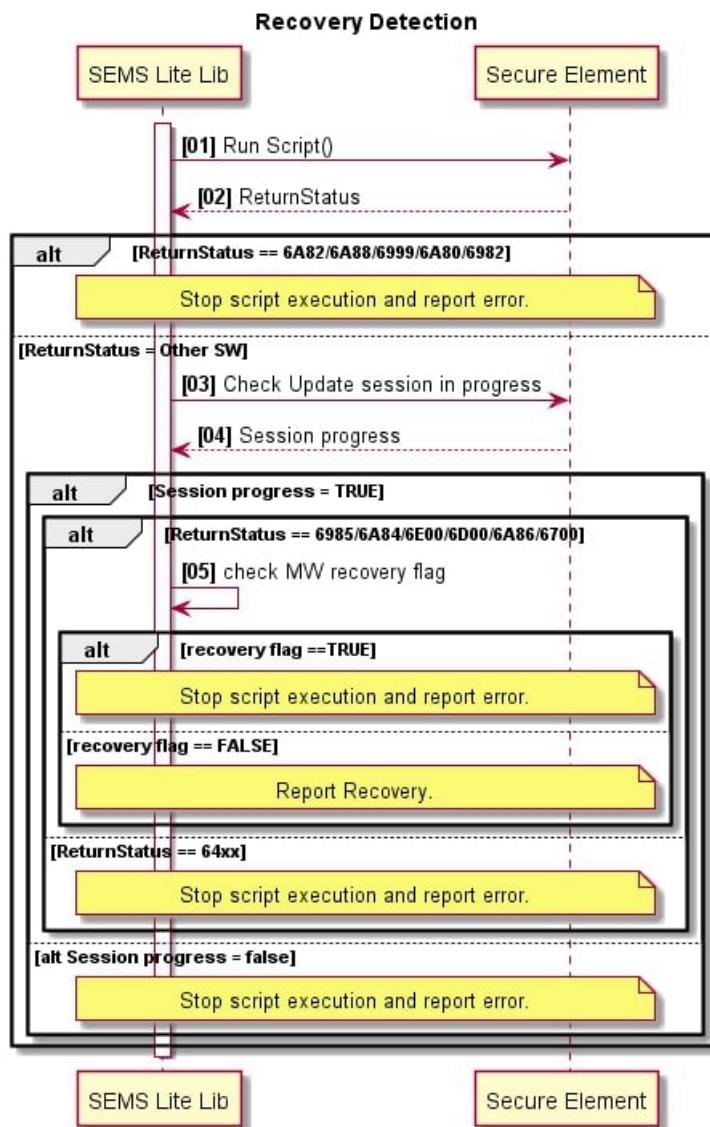
In case the upgrade can not complete, user would receive return code “Do Recovery”. User must load recovery script to resume SE to old Applet.

1. Call `sems_lite_agent_init_context()`, `sems_lite_agent_session_open()` and `sems_lite_agent_load_package()` as those in [Case 1: Basic Usage](#).
2. SEMS Lite agent would query tear status from SE as in [Case 2: In case there was a tearing](#).
3. SEMS Lite agent load the script from package to SE. SE can't complete upgrade and inform SEMS Lite agent to do recovery by response.
4. SEMS Lite agent checking the response and return “DoRecovery” to user.
5. User must find the recovery script which is used to load the old Applet.
6. User load the recovery script and get response from SEMS Lite agent.



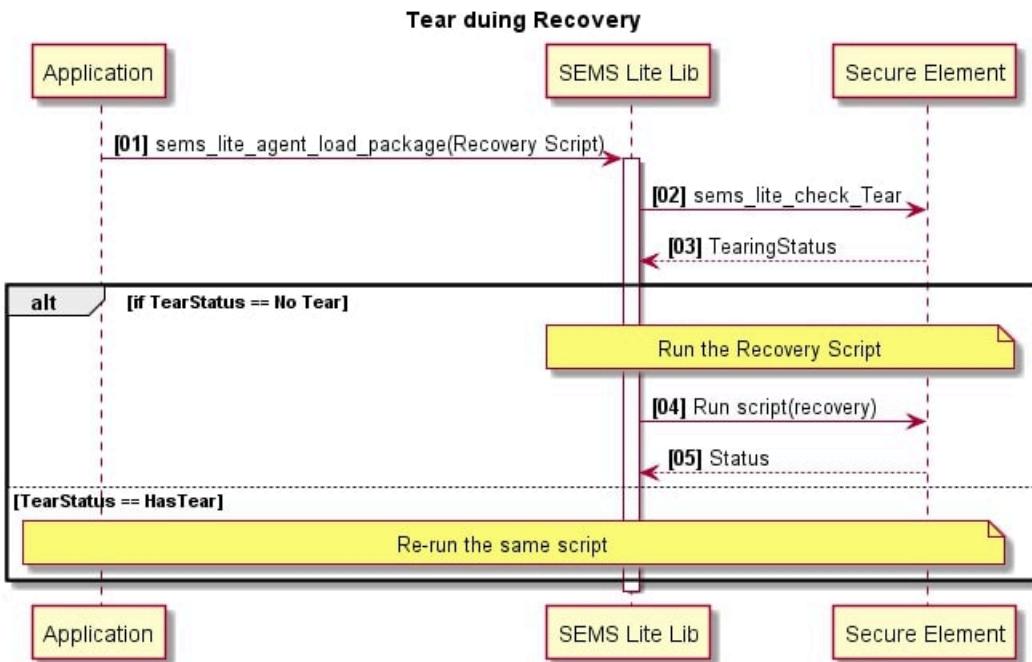
Details for SE to report “Do Recovery” can be found in following:





7.3.4 Case 4: Tearing during recovery

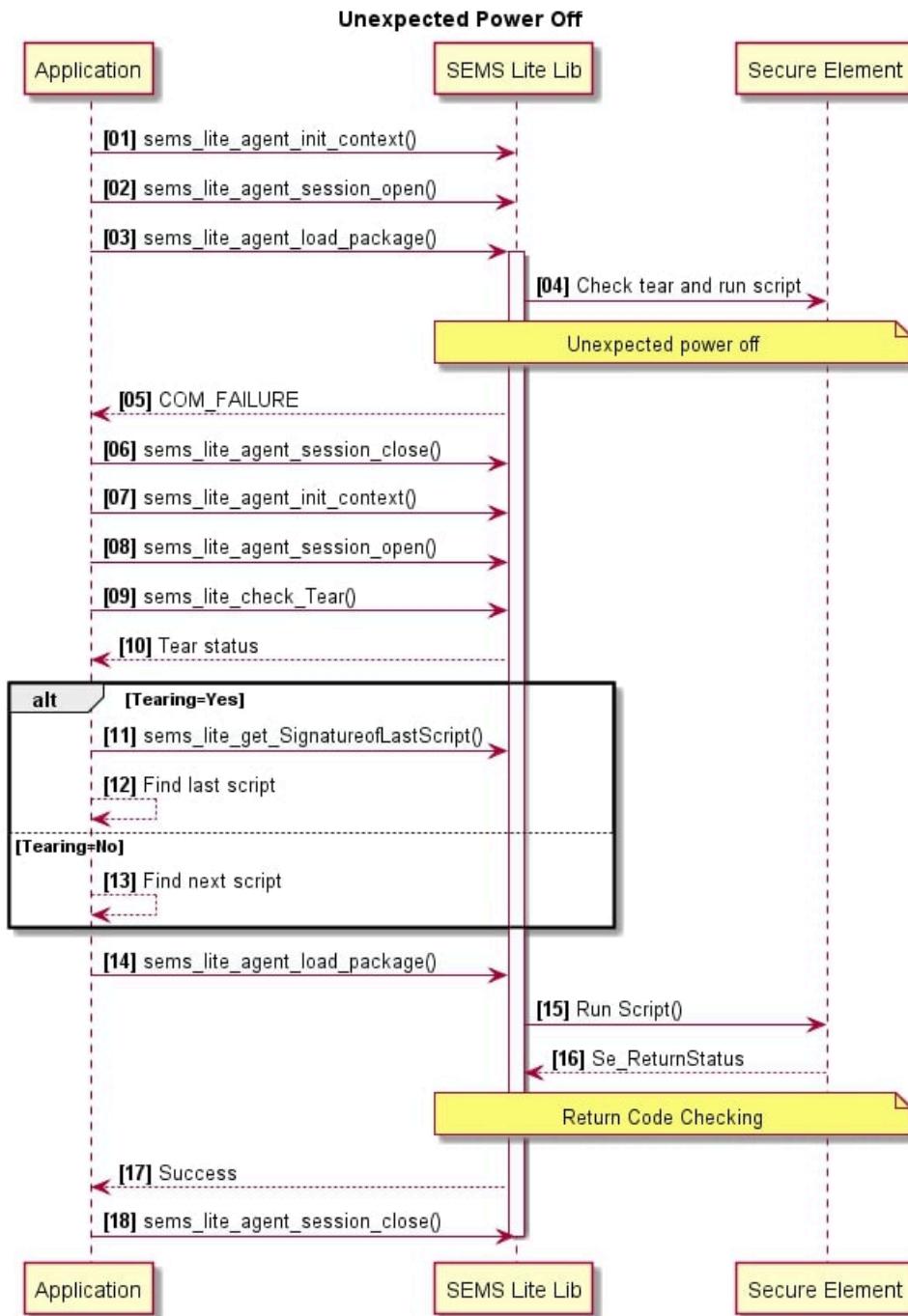
Case 4: In case tear happens when loading recovery package, User should re-run the script in the same way described in [Case 2: In case there was a tearing.](#)



7.3.5 Case 5: Load Key rotate script encounter SE unexpected power off

In case the SE is powered off unexpectedly during uploading key rotate script, user would receive return code "COM_FAILURE" which means communication failure. User has 2 choices:

1. Load next script. SEMS Lite agent will check tear status and inform user if it's required to re-run last script. It's same to [Case 2: In case there was a tearing](#).
2. User software checks tear status and then decides whether to re-load the broken script as following:
 1. Call `sems_lite_agent_init_context()` and `sems_lite_agent_session_open()` as those in [Case 1: Basic Usage](#).
 2. Call `sems_lite_check_Tear()` to decide if tear happens for last upgrade.
 3. In case tear happens, call `sems_lite_get_SignatureofLastScript()` to get last script signature
 4. Find the correct script by signature and load the tear script again with `sems_lite_agent_load_package()`.



7.3.6 API Calls for Usage of SEMS Lite Agent

```
#include "sems_lite_api.h"
```

Declare a SEMS Lite input buffer: SEMS Lite input buffer is hex file converted from protobuf file

Declare SEMS Lite context: SEMS Lite context holds flags and variables used through the whole loading process

```
sems_lite_agent_ctx_t s_sems_lite_agent_ctx = {0};
```

Initialize the context:

```
rv = sems_lite_agent_init_context(&s_sems_lite_agent_ctx, &pCtx->session);
```

Open session:

```
rv = sems_lite_agent_session_open(&s_sems_lite_agent_ctx);
```

Load Package:

```
status = sems_lite_agent_load_package(  
    &s_sems_lite_agent_ctx, &multicast_package);
```

Close the session:

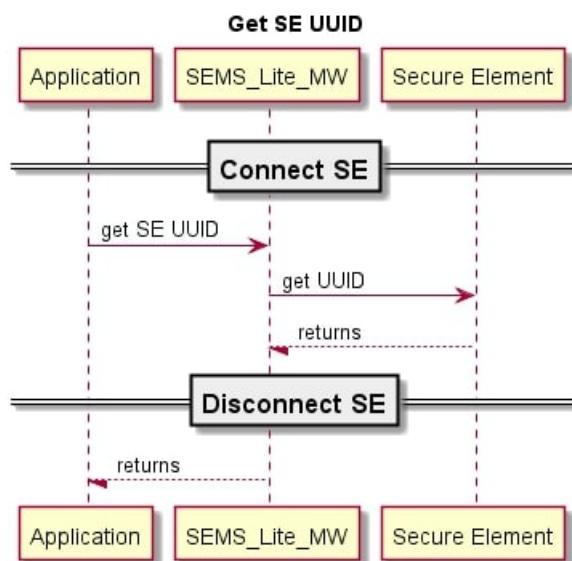
```
rv = sems_lite_agent_session_close(&s_sems_lite_agent_ctx);
```

7.4 SEMS Lite management APIs

SEMS Lite management API request data to SE. ... TODO To be received from SEMS Lite Applet interface documentation.

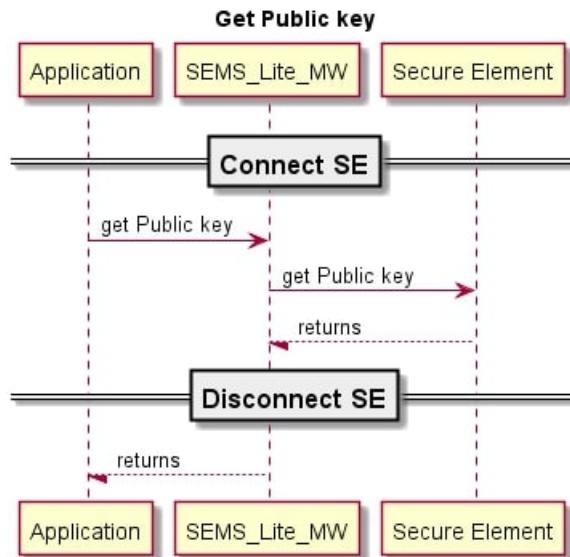
7.4.1 Get UUID

See [sems_lite_get_UUID](#)



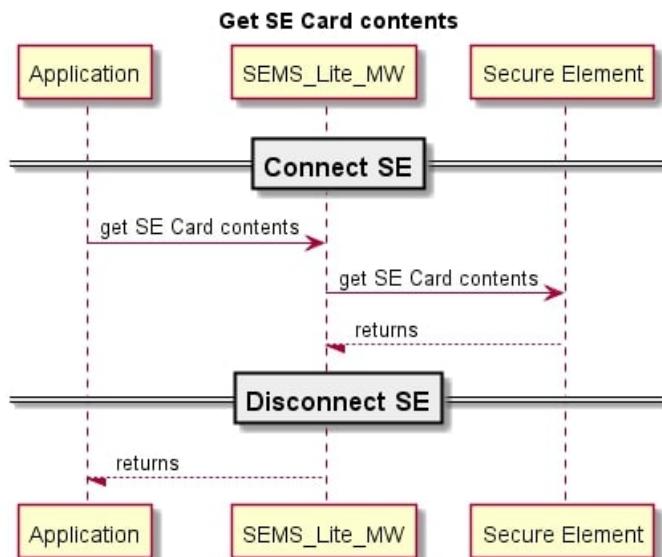
7.4.2 Get Public Key

See [sems_lite_get_Publickey](#)



7.4.3 Get Card contents

- See [sems_lite_get_SEAppInfoRAW](#)
- See [sems_lite_get_SEAppInfo](#)
- See [sems_lite_get_SEPkgInfoRAW](#)
- See [sems_lite_get_SEPkgInfo](#)



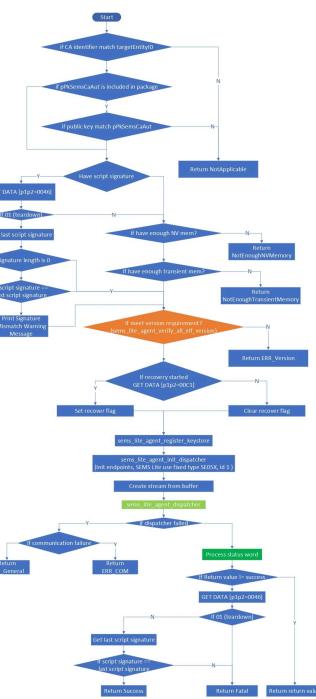
7.5 SEMS Lite Agent Package Load Process

When SE update package is loaded to SEMS Lite agent, SEMS Lite agent executes following steps to load package (offline) to the SE.

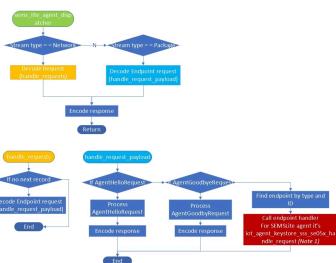
1. Check if target entity ID match.
2. Check tear status which indicate if last load operation succeed. If not, SEMS Lite agent will indicate user to re-run the tear script.
3. Check if recovery has started and record flag accordingly. This flag is used later in error code handling.
4. Check if meet minimum previous version and version request.
5. Decode the protobuf stream to find valid SEMS Lite agent package.
6. If the package includes host command, take relative actions (reset SE, etc).
7. If the package includes APDU command, send the APDU command to SE.
8. Get the status word from SE. Tranlate the status word to customer friendly return value.
9. Check tear status which indicate if this load operation succeed. If not, SEMS Lite agent will indicate user to re-run the tear script.

Refer to following flowchart for detail:

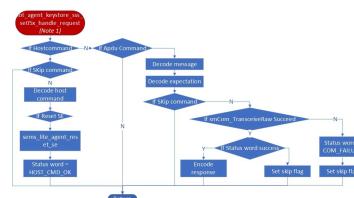
- **Flowchart Part 1:**



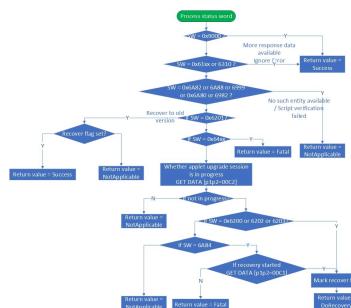
- **Flowchart Part 2:**



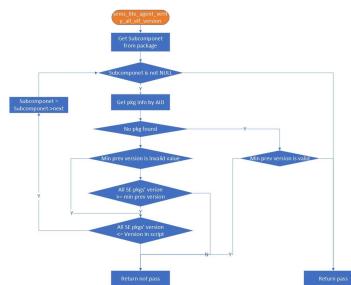
- Flowchart Part 3:



- Flowchart Part 4:



- Flowchart Part 5:



Note 1: SEMS Lite agent and iot hub module share protobuf stream processing function. In this function, several keystore endpoints could attach to the dispatcher. But in SEMS Lite agent case, there is only one keystore endpoint which is sems lite agent.

7.6 SEMSLite Types and APIs

- [SEMSLite Agent Types](#)
- [SEMSLite Agent APIs](#)

7.6.1 SEMSLite Agent Types

groupsems_lite_agent_types

SEMSLite Types to be used with SEMSLite APIs.

Typedefs

typedef [sems_lite_SEAppInfoList_t](#) [sems_lite_PKGInfoList_t](#)

Same as [sems_lite_SEAppInfoList_t](#) for list of installed packages

Enums

enum [_sems_lite_recovery_status_t](#)

Indicates the current state of recovery process in a System

Values:

enumerator [sems_lite_recovery_not_started](#)

Recovery Not Started

enumerator [sems_lite_recovery_started](#)

Recovery Started

enum [_sems_lite_status](#)

Status of SEMS Lite update operation

Values:

enumerator [kStatus_SEMS_Lite_Success](#)

Operation was successful

enumerator [kStatus_SEMS_Lite_ERR_COM](#)

Communication Error

enumerator [kStatus_SEMS_Lite_ERR_DoReRun](#)

Update not completed please provide update package again.

enumerator [kStatus_SEMS_Lite_ERR_NotApplicable](#)

Update not applicable on this Chip/type.

enumerator [kStatus_SEMS_Lite_ERR_DoRecovery](#)

Update can not be completed. Please provide recovery package, to roll back to last working version.

enumerator [kStatus_SEMS_Lite_ERR_Fatal](#)

Unresolvable error.

enumerator [kStatus_SEMS_Lite_ERR_NotEnoughNVMemory](#)

Not enough NV memory.

enumerator [kStatus_SEMS_Lite_ERR_NotEnoughTransientMemory](#)

Not enough transient memory.

enumerator [kStatus_SEMS_Lite_ERR_MinPreviousVersion](#)

Current SE version not meet min previous version request.

```
enumerator kStatus_SEMS_Lite_ERR_OlderVersion
    Older than current SE version
enumerator kStatus_SEMS_Lite_ERR_General
    General error.

enum _sems_lite_tearDown_status_t
    Check Tear down Amd-I section 4.14
    Values:
        enumerator sems_lite_notear
            The script has been completely executed
        enumerator sems_lite_tear
            Script execution was interrupted because of teardown.
        enumerator sems_lite_tearDown_status_invalid

enum _sems_lite_upgradeProgress_status_t
    Track the progress of an upgrade process in a software system, distinguishing between when the upgrade session is active or not, and providing a special value for invalid states.
    Values:
        enumerator sems_lite_upgrade_not_inProgress
            Upgrade session Not in Progress
        enumerator sems_lite_upgrade_inProgress
            Upgrade session In Progress
        enumerator sems_lite_upgrade_invalid

enum _sems_lite_version_check_result_t
    Status of SEMS Lite version check result
    Values:
        enumerator kStatus_SEMS_Lite_Version_Pass
            Operation was successful
        enumerator kStatus_SEMS_Lite_Version_ERR_MIN
            Failed due to min previous version.
        enumerator kStatus_SEMS_Lite_Version_ERR_Downgrade
            Failed due to downgrade.

struct _sems_lite_available_mem_t
    #include <sems_lite_api.h>
    Represents available memory for different components , such as COD,COR PersistentMemory, IDX

Public Members
    uint32_t availableCODMemory
        Indicates the available memory for COD (Card Operating System)
    uint32_t availableCORMemory
        Indicates the available memory for COR (Card operating Runtime)
    uint32_t availableIDX
        Indicates Available IDX
```

uint32_t availablePersistentMemory

Indicates Available PersistentMemory

uint32_t freePHeapCentralGap

Indicates amount of free memory in the central gap of the PHeap

uint32_t freeTransient

Represents free transient memory from cOD from JCOP

struct _sub_component_metaData_t**Public Members****size_t aidLen**

Length of aid

uint8_t minimumPreviousVersion[2]

Minimum version number of this subcomponent as installed on the secure element before this script is executed. If this field is omitted there is no minimum version requirement, e.g. initial Installation of an applet. If the minimumPreviousVersion is 4.11 but the secure element contains version 3.2, the intermediate update packages need to be applied to reach version 4.11 before this package. Version information major.minor, according to java card virtual machine package versions, both values in range 0 to 127. byte 0 is Major value, byte 1 is Minor value.

size_t nameLen

Length of name

const uint8_t *pAid

The Application Identifier (AID) of the Executable Load File (ELF) which is in this subcomponent. This is stored as string to have it formatted in hexadecimal and therefore recognizable form. Encoded as binary.

char *pName

A human readable name for this subcomponent.

struct _sub_component_metaData_t *pNextSubComponentMetaData

Next subcomponent meta data

char *pSWContentRegister

NXP Software Content Register, specifies the Outgoing License, mostly: LA_OPT_NXP_Software_License.

size_t swContentRegisterLen

Length of SWContentRegister

uint8_t version[2]

Version information of this subcomponent. Version information major.minor, according to java card virtual machine package versions, both values in range 0 to 127. byte 0 is Major value, byte 1 is Minor value.

struct multicast_package_t

#include <sems_lite_api.h>

Information about package

Public Members**size_t multicastCommandsLen**

Length of multicast commands (as pointed by pMulticastCommands). If multicastCommandsLen mismatches to contents of pMulticastCommands, there will be undesired behavior of system.

size_t multicastPackageNameLen

Length of multicast Package name

uint8_t multicastPackageVersion[2]

Version information of this MulticastPackage, describing the sum version over all contained content. In case exactly one SubComponent is contained this version is equal to the version of this SubComponent. Version information major.minor, both values in range 0 to 127. byte 0 is Major value, byte 1 is Minor value.

size_t pkSemsCaAutLen

Length of pPkSemsCaAut

const uint8_t *pMulticastCommands

The complete Multicast Applet Loader Script (certificate, signature, encrypted and signed commands) in protobuf format.

const char *pMulticastPackageName

Giving a descriptive name to the content of the Multicast Package. It is possible that one Multicast Package contains multiple Applet binaries and also that a Multicast Package contains no Applet binary at all (e.g. for deletion).

const uint8_t *pPkSemsCaAut

The PkSemsCaAut key found in the SEMS_SP_AUT.txt file as PK.AP.AUT.

const uint8_t *pSignatureOverCommands

The signature over the multicast commands in an machine readable form. So it does not have to be parsed from the script commands. String encoding is chosen here, as mayna json parsers can not handle such large integer values.

const sub_component_metaData_t *pSubComponentMetaData

A list of subcomponents of this MulticastPackage, designating all Executable Load Files (ELFs) Contained. It usually contains one entry, but can have multiple in the case multiple dependent ELF's get modified. This list can be empty, e.g. for a KeyRotation or content deletion MulticastPackage.

uint32_t requiredFreeBytesNonVolatileMemory

Minimum required free Non Volatile memory in bytes that have to be free on the target device before execution of MulticastPackage.

uint32_t requiredFreeBytesTransientMemory

Minimum required free transient (RAM) memory in bytes that have to be present on the target device before execution of MulticastPackage.

uint32_t semsLiteAPIVersion

Version information of SEMS Lite Agent API Defined follow SEMS_LITE_AGENT_VER_MAJOR

size_t signatureOverCommandsLen

Length of signatureOverCommands

uint8_t target12Nc[6]

Target 12nc is a 12 digit numerical code identifying the target device where this MulticastPackage is intended to be executed on, as known to customers and used on EdgeLock2Go to identify device types.

uint8_t targetEntityID[16]

Entity ID, 16bytes long Binary Coded Decimal, of the target device where this MulticastPackage is intended to be executed on. It is an identifier of the key-set of the Multicast Applet Loader. As the key-set and related Entity ID can change over the lifetime of the device it is separated from the Commercial Name. This is stored to have it formatted in the BCD encoded and therefore recognizable form.

struct sems_lite_agent_ctx_t

```
#include <sems_lite_agent_context.h>
```

This structure seems to manage the context and state of a communication session, possibly involving interactions with multiple applets or components within a system.

Public Members

uint32_t freePHeapCentralGap

Indicates the amount of free memory in the central gap of the PHeap

uint32_t freeTransient

Represents free transient memory information, potentially related to COD (Card Operating System) information from JCOP.

Se05xSession_t *pS05x_Ctx

Boot context

bool recovery_executed

Flag for get recovery script

bool session_opened

Flag for session status: Open/Closed

bool skip_next_commands

Skip following commands

Internal state variable.

When one APDU in this APDU stream has failed, skip through all the next APDUs (do not send them) and return back to the caller.

uint32_t status_word

Status word of last R-APDU.

struct **sems_lite_agent_request_ctx_t**

#include <sems_lite_agent_context.h>

This structure likely facilitates managing the state and progress of requests made by the SEMS Lite agent, including keeping track of the current command being processed, checkpoints for managing progress, and retry counts for error handling.

Public Members

int checkpoint_index

index of the checkpoint, which is a specific point in the sequence of commands that marks progress or a known state.

int current_command_index

index of the current command in the sequence of commands being processed.

int retry_count

number of times a command has been retried in case of failure.

struct **sems_lite_SEAppInfoList_t**

#include <sems_lite_api.h>

Information of about Applet/Package

See Table 11-36: GlobalPlatform Registry Data (TLV), GPCardSpc_v2.2.pdf

The response from Applet is put to rspBuf

After parsing that response, the pointers to respective members is set and it points to relevant part in rspBuf, this way saving memory. However, the Length is updated so that application use this information.

Public Members

uint8_t AIDLen

Length of the Applet ID
uint8_t LifeCycleState
Life-cycle state
uint8_t LoadFileAIDLen
Length of LoadFileAID.
uint8_t LoadFileVersionNumberLen
Length of pLoadFileVersionNumber.
uint8_t *pAID
Applet ID
uint8_t *pLoadFileAID
Application's Executable Load File AID.
uint8_t *pLoadFileVersionNumber
Executable Load File Version Number.
uint8_t *pPrivileges
Privileges.
uint8_t PrivilegesLen
Length of Privileges
uint8_t *pSecurityDomainAID
Associated Security Domain's AID.
uint8_t rspBuf[(256 + 5)]
Response from Applet.
size_t rspBufLen
Length of response from Applet.
uint8_t SecurityDomainAIDLen
Length of SecurityDomainAID.

7.6.2 SEMSLite Agent APIs

groupsems_lite_agent

API to load an available update package on the SE.

Functions

sss_status_t sems_lite_agent_init_context([sems_lite_agent_ctx_t](#) *context, [sss_session_t](#) *boot_ctx)

Initialize SEMS Lite agent context.

This function is used to initialize SEMS Lite agent context.

Parameters

- **context** – Pointer to sems lite agent context.
- **boot_ctx** – Pointer to sss session context

Return values

- **kStatus_SEMS_Lite_Success** – The operation has completed successfully.
- **kStatus_SEMS_Lite_ERR_General** – The operation has failed.

Returns

Status of the operation

`sems_lite_status_t sems_lite_agent_load_package(sems_lite_agent_ctx_t *context, multicast_package_t *multiPkgBuf)`

Load Applet package.

This function load an available update package on the SE and assure the tearing safe update of the SE.

Note:

More return codes would be added to request host to either retry or install older package.

Parameters

- `context` – Pointer to sems lite agent context.
- `multiPkgBuf` – Pointer to package. It must follow the format defined in [multicast_package_t](#).

Return values

- `kStatus_SEMS_Lite_Success` – The operation has completed successfully.
- `kStatus_SEMS_Lite_ERR_COM` – Communication to SE failed.
- `kStatus_SEMS_Lite_ERR_DoReRun` – Update not completed please provide update package again.
- `kStatus_SEMS_Lite_ERR_NotApplicable` – Update not applicable on this Chip/type.
- `kStatus_SEMS_Lite_ERR_DoRecovery` – Update can not be completed. Please provide recovery package, to roll back to last working version.
- `kStatus_SEMS_Lite_ERR_Fatal` – Unresolvable error. (This category of errors can only appear in testing of an update package, in the case of NXP updates this is testes before by NXP)
- `kStatus_SEMS_Lite_ERR_NotEnoughNVMemory` – Don't have has enough NV memory for the SEMS Lite Script.
- `kStatus_SEMS_Lite_ERR_NotEnoughTransientMemory` – Don't have enough transient memory for the SEMS Lite Script .

Returns

Status of the operation

`sss_status_t sems_lite_agent_session_close(sems_lite_agent_ctx_t *context)`

Close the connection to SEMS Lite Applet.

Parameters

`context` – The context

Return values

- `kStatus_SEMS_Lite_Success` – Could close connection 1.
- `kStatus_SEMS_Lite_ERR_General` – Could not close connection 1.

Returns

The api status.

`sss_status_t sems_lite_agent_session_open(sems_lite_agent_ctx_t *context)`

Open a Physical connection to SEMS Lite Applet.

Calling this API opens Locical Connection 1 and selecs the SEMS Lite applet.

Parameters

`context` – SEMS Lite Agent Context

Return values

- **kStatus_SEMS_Lite_Success** – Could connect to SEMS Lite Applet.
- **kStatus_SEMS_Lite_ERR_General** – Could not connect to SEMS Lite Applet.

Returns

The api status.

[sss_status_t sems_lite_check_AppletRecoveryStatus\(sems_lite_agent_ctx_t *pContext, sems_lite_recovery_status_t *pRecoveryStatus\)](#)

Check Applet Recovery Status.

This API will return the status of applet recovery status

Parameters

- **pContext** – Pointer to sems lite agent context.
- **pRecoveryStatus** – Pointer to recovery status.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

[sss_status_t sems_lite_check_AppletUpgradeProgress\(sems_lite_agent_ctx_t *pContext, sems_lite_upgrade_progress_status_t *pUpgradeStatus\)](#)

Check Applet Upgrade Progress.

This API will return the status of applet upgrade progress status

Parameters

- **pContext** – Pointer to sems lite agent context.
- **pUpgradeStatus** – Pointer to upgrade status.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

[sss_status_t sems_lite_check_Tear\(sems_lite_agent_ctx_t *pContext, sems_lite_teardown_status_t *pTearStatus\)](#)

Check Tear during script execution.

This API will check whether there was tear during script execution

Parameters

- **pContext** – Pointer to sems lite agent context.
- **pTearStatus** – Pointer to tear status.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

[sss_status_t sems_lite_get_AgentVersion\(uint8_t *pRspBuf, size_t *pRspBufLen\)](#)

Get the SEMS Lite Agent version.

This API will return the SEMS Lite Agent Version no.

Parameters

- **pRspBuf** – Pointer to response Buffer.
- **pRspBufLen** – Pointer to length of the response Buffer

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

```
sss_status_t sems_lite_get_AppletVersion(sems_lite_agent_ctx_t *pContext, uint8_t *pRspBuf, size_t *pRspBufLen)
```

Get the Applet version.

This API will return the SEMS Lite Applet Version no.

Parameters

- **pContext** – Pointer to sems lite agent context.
- **pRspBuf** – Pointer to response Buffer.
- **pRspBufLen** – Pointer to length of the response Buffer

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

```
sss_status_t sems_lite_get_available_mem(sems_lite_agent_ctx_t *pContext, uint8_t *pAvailableMem)
```

```
{ function_description }
```

Parameters

- **pContext** – The context
- **pAvailableMem** – Available Memory Space Information

Returns

The sss status.

```
sss_status_t sems_lite_get_CA_identifier(sems_lite_agent_ctx_t *pContext, uint8_t *pRspBuf, size_t *pRspBufLen)
```

Get the CA Identifier.

This API will return the CA Identifier.

Parameters

- **pContext** – Pointer to sems lite agent context.
- **pRspBuf** – Pointer to response Buffer.
- **pRspBufLen** – Pointer to length of the response Buffer

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

`sss_status_t sems_lite_get_ENCIdentifier(sems_lite_agent_ctx_t *pContext, uint8_t *pRspBuf, size_t *pRspBufLen)`

Get the ENC Identifier.

This API will return the ENC Identifier.

Parameters

- `pContext` – Pointer to sems lite agent context.
- `pRspBuf` – Pointer to response Buffer.
- `pRspBufLen` – Pointer to length of the response Buffer

Return values

- `kStatus_SSS_Success` – The operation has completed successfully.
- `kStatus_SSS_Fail` – The operation has failed.

Returns

Status of the operation

`sss_status_t sems_lite_get_FIPS_EC_parameter_type(sems_lite_agent_ctx_t *pContext, uint8_t *pParamType)`

Get Configured EC domain parameter type.

This API will return Configured EC domain parameter type

Parameters

- `pContext` – Pointer to sems lite agent context.
- `pParamType` – Pointer to parameter type.

Return values

- `kStatus_SSS_Success` – The operation has completed successfully.
- `kStatus_SSS_Fail` – The operation has failed.

Returns

Status of the operation

`sss_status_t sems_lite_get_FIPS_info(sems_lite_agent_ctx_t *pContext, uint8_t *pFIPSInfo)`

Get Configured FIPS Information.

This API will return Configured FIPS Information

Parameters

- `pContext` – Pointer to sems lite agent context.
- `pFIPSInfo` – Pointer to FIPS Info.

Return values

- `kStatus_SSS_Success` – The operation has completed successfully.
- `kStatus_SSS_Fail` – The operation has failed.

Returns

Status of the operation

`sss_status_t sems_lite_get_Publickey(sems_lite_agent_ctx_t *pContext, uint8_t *pRspBuf, size_t *pRspBufLen)`

Read Public Key.

This API will read root certificates public key of the device.

Parameters

- **pContext** – Pointer to sems lite agent context.
- **pRspBuf** – Pointer to response Buffer.
- **pRspBufLen** – Pointer to length of the response Buffer.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

sss_status_t **sems_lite_get_SEAppInfo**(**sems_lite_agent_ctx_t** ***pContext**, **const uint8_t** ***searchAID**, **uint8_t** **searchAidLen**, **sems_lite_SEAppInfoList_t** ***pAppInfo**, **size_t** ***pAppInfoLen**)

Low level API to get App INFO from the SE according to format mentioned in - Table 11-36: GlobalPlatform Registry Data (TLV), GPCardSpc_v2.2.pdf.

This API will read the currently present Applications ELF/ELM AIDs and versions as well as the present instances from the SE.

Parameters

- **pContext** – Pointer to sems lite agent context.
- **searchAID** – **[in]** The search aid
- **searchAidLen** – **[in]** The search aid length
- **pAppInfo** – Parsed structures
- **pAppInfoLen** – **[inout]** Length of parsed structures.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

sss_status_t **sems_lite_get_SEAppInfoRAW**(**sems_lite_agent_ctx_t** ***pContext**, **const uint8_t** ***searchAID**, **uint8_t** **searchAidLen**, **uint8_t** ***pRspBuf**, **size_t** ***pRspBufLen**)

Low level API to get Raw App INFO from the SE.

This API will read the currently present Applications ELF/ELM AIDs and versions as well as the present instances from the SE.

Parameters

- **pContext** – Pointer to sems lite agent context.
- **pRspBuf** – **[out]** Pointer to response Buffer.
- **searchAID** – **[in]** The search aid
- **searchAidLen** – **[in]** The search aid length
- **pRspBufLen** – **[inout]** Pointer to length of the response Buffer.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

sss_status_t **sems_lite_get_SEPkInfo**(**sems_lite_agent_ctx_t** ***pContext**, **const uint8_t** ***searchAID**, **uint8_t** **searchAidLen**, **sems_lite_SEAppInfoList_t** ***pAppInfo**, **size_t** ***pAppInfoLen**)

Low level API to get PKG INFO from the SE according to format mentioned in - Table 11-36: GlobalPlatform Registry Data (TLV), GPCardSpc_v2.2.pdf.

This API will read the currently present Applications ELF/ELM AIDs and versions as well as the present instances from the SE.

Parameters

- **pContext** – Pointer to sems lite agent context.
- **searchAID** – **[in]** The search aid
- **searchAidLen** – **[in]** The search aid length
- **pAppInfo** – Parsed structures
- **pAppInfoLen** – **[inout]** Length of parsed structures.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

`sss_status_t sems_lite_get_SEPkInfoRAW(sems_lite_agent_ctx_t *pContext, const uint8_t *searchAID, uint8_t searchAidLen, uint8_t *pRspBuf, size_t *pRspBufLen)`

Low level API to get RAW PKG INFO from the SE.

This API will read the currently present Applications ELF/ELM AIDs and versions as well as the present instances from the SE.

Parameters

- **pContext** – Pointer to sems lite agent context.
- **searchAID** – **[in]** The search aid
- **searchAidLen** – **[in]** The search aid length
- **pRspBuf** – **[out]** Pointer to response Buffer.
- **pRspBufLen** – **[inout]** Pointer to length of the response Buffer.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

`sss_status_t sems_lite_get_SignatureofLastScript(sems_lite_agent_ctx_t *pContext, uint8_t *pRspBuf, size_t *pRspBufLen)`

Get the signature of last executed script.

This API will called in case there is tear down

Parameters

- **pContext** – Pointer to sems lite agent context.
- **pRspBuf** – Pointer to response Buffer.
- **pRspBufLen** – Pointer to length of the response Buffer

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

`sss_status_t sems_lite_get_UUID(sems_lite_agent_ctx_t *pContext, uint8_t *pRspBuf, size_t *pRspBufLen)`

Retrieve UUID from SE.

This API read UUID of the SE.

Parameters

- `pContext` – Pointer to sems lite agent context.
- `pRspBuf` – Pointer to response Buffer.
- `pRspBufLen` – Pointer to length of the response Buffer.

Return values

- `kStatus_SSS_Success` – The operation has completed successfully.
- `kStatus_SSS_Fail` – The operation has failed.

Returns

Status of the operation

7.7 SEMS Lite Known Issue

7.7.1 Signature for rerun script is incorrect occasionally

- Issue description: In case of tear (e.g. unexpected power off when upgrading SE), when user tries to load new script, SEMS Lite agent will only allow the re-run script. This script should be the one that is interrupted. But in special case of very early tearing, SEMS Lite agent may expect the user to re-run the script that's already existed in SE before upgrade start.
- Issue workaround: Using `sems_lite_get_SignatureofLastScript()` to get the signature of the expected script. Find the script and load it.

7.8 SEMS Lite DEMOs

The DEMOs applicable to SEMS Lite are listed at [Section 5.1.1.11SEMS Demos](#)

8 Plugins / Add-ins

Plugins / Add-ins for other platforms

See also [Android: Hikey960](#)

8.1 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, ...) the following functionality can be made available over the OpenSSL Engine interface:

- EC crypto
 - EC sign/verify
 - ECDH compute key
 - Montgomery ECDH
- RSA crypto
 - RSA sign/verify
 - RSA priv_key_decrypt/pub_key_encrypt

- Fetching random data

8.1.1 General

8.1.1.1 OpenSSL versions

The OpenSSL Engine is compatible with OpenSSL versions 1.0.2 or 1.1.1.

8.1.1.2 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, SE051, or SE052). The `demos/linux/common` folder of this SW package contains 4 reference configuration files covering SE050 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.

8.1.1.3 Platforms

The OpenSSL engine can be used on iMX boards (running Linux) or on Raspberry Pi (running Raspbian).

8.1.2 Keys

8.1.2.1 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. If the input key is not reference key, execution will roll back to openssl software implementation

8.1.2.2 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 data structure 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:
    00:00:00:7D: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.*

Note:

- *For EC Montgomery curves, openssl allows only the private key to be set. So the reference key created will not have the valid public key.*

8.1.2.3 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:
  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.*

8.1.3 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

8.1.4 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:

8.1.4.1 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
```

8.1.4.2 openssl_provisionEC.py

usage: openssl_provisionEC.py [-h] --key_type KEY_TYPE

[--connection_type CONNECTION_TYPE] [--connection_data CONNECTION_DATA] [--subsystem SUBSYSTEM] [--auth_type AUTH_TYPE] [--scpkey SCPKEY]

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, brainpool P192r1, brainpoolP224r1, brainpool P256r1, brainpoolP320r1, brainpool P384r1, brainpoolP512r1, secp160k1, secp192k1, secp224k1, secp256k1
---------------------	--

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
--auth_type AUTH_TYPE	Supported subsystem => None, PlatformSCP, UserID, ECKey, AESKey. Default: None

-scpkey SCPKEY

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
```

8.1.4.3 openssl_EccSign.py

usage: openssl_EccSign.py [-h] --key_type KEY_TYPE
[--connection_data CONNECTION_DATA] [--disable_sha1 DISABLE_SHA1]

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, brainpool P192r1, brainpoolP224r1, brainpool P256r1, brainpoolP320r1, brainpool P384r1, 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
--disable_sha1 DISABLE_SHA1	Parameter to disable SHA1 => eg. True, False. Default: False

Example invocation:

```
python openssl_EccSign.py --key_type prime256v1
python openssl_EccSign.py --key_type secp160k1 --connection_data 127.0.0.1:8050
```

8.1.4.4 openssl_Ecdh.py

usage: openssl_Ecdh.py [-h] --key_type KEY_TYPE

[--connection_data CONNECTION_DATA] [--disable_sha1 DISABLE_SHA1]

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, brainpool P192r1, brainpoolP224r1, brainpool P256r1, brainpoolP320r1, brainpool P384r1, 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
--disable_sha1 DISABLE_SHA1	Parameter to disable SHA1 => eg. True, False. Default: False

Example invocation:

```
python openssl_Ecdh.py --key_type prime256v1
```

```
python openssl_Ecdh.py --key_type secp160k1 --connection_data 127.0.0.1:8050
```

8.1.4.5 ecc_all.py

usage: ecc_all.py [-h] [--connection_type CONNECTION_TYPE]

[--connection_data CONNECTION_DATA] [--subsystem SUBSYSTEM] [--auth_type AUTH_TYPE] [--scpkey SCPKEY] [--disable_sha1 DISABLE_SHA1] [--fips FIPS]

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 => se051, se050. Default: se050
--auth_type AUTH_TYPE	Supported subsystem => None, PlatformSCP, UserID, ECKey, AESKey. Default: None

--scpkey SCPKEY --disable_sha1 DISABLE_SHA1

Parameter to disable SHA1 => eg. True, False. Default: False

--fips FIPS	FIPS Testing => eg. True, False. Default: False
-------------	---

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
```

8.1.4.6 openssl_provisionRSA.py

usage: openssl_provisionRSA.py [-h] --key_type KEY_TYPE

[--connection_type CONNECTION_TYPE] [--connection_data CONNECTION_DATA] [--subsystem SUBSYSTEM] [--auth_type AUTH_TYPE] [--scpkey SCPKEY]

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
--auth_type AUTH_TYPE	Supported subsystem => None, PlatformSCP, UserID, ECKey, AESKey. Default: None

-scpkey SCPKEY

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
```

8.1.4.7 openssl_RSA.py

usage: openssl_RSA.py [-h] --key_type KEY_TYPE
[--connection_data CONNECTION_DATA] [--disable_sha1 DISABLE_SHA1]

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
--disable_sha1 DISABLE_SHA1	Parameter to disable SHA1 => eg. True, False. Default: False

Example invocation:

```
python openssl_RSA.py --key_type rsa2048
python openssl_RSA.py --key_type rsa4096 --connection_data 127.0.0.1:8050
```

8.1.4.8 rsa_all.py

usage: rsa_all.py [-h] [--connection_data CONNECTION_DATA]

[--connection_type CONNECTION_TYPE] [-subsystem SUBSYSTEM] [-auth_type AUTH_TYPE] [-scpkey SCPKEY] [-disable_sha1 DISABLE_SHA1] [-fips FIPS]

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 => t1oi2c, sci2c, vcom, jrcpv1, jrcpv2, pcsc. Default: t1oi2c
--subsystem SUBSYSTEM	Supported subsystem => se050. Default: se050
--auth_type AUTH_TYPE	Supported subsystem => None, PlatformSCP, UserID, ECKey, AESKey. Default: None

-scpkey SCPKEY -disable_sha1 DISABLE_SHA1

Parameter to disable SHA1 => eg. True, False. Default: False

--fips FIPS	FIPS Testing => eg. True, False. Default: False
-------------	---

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
```

8.1.4.9 openssl_provisionEC_mont.py

usage: openssl_provisionEC_mont.py [-h] --key_type KEY_TYPE

```
[--connection_type CONNECTION_TYPE] [--connection_data CONNECTION_DATA] [--subsystem SUBSYSTEM] [--auth_type AUTH_TYPE] [-scpkey SCPKEY]
```

Provision attached secure element with EC montgomery keys

This example generates EC montgomery key files (.pem) (existing ones overwritten). Performs debug reset of the attached secure element. Attached secure element provisioned with EC montgomery key. Creates reference key from the injected EC montgomery key.

optional arguments:

-h --help	show this help message and exit
--------------	---------------------------------

required arguments:

--key_type KEY_TYPE	Supported key types => x25519, x448
---------------------	-------------------------------------

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
--auth_type AUTH_TYPE	Supported subsystem => None, PlatformSCP, UserID, ECKey, AESKey. Default: None

-scpkey SCPKEY

Example invocation:

```
python openssl_provisionEC_mont.py --key_type x25519
python openssl_provisionEC_mont.py --key_type x25519 --connection_data
169.254.0.1:8050
python openssl_provisionEC_mont.py --key_type x448 --connection_type jrcpv2 --
connection_data 127.0.0.1:8050
python openssl_provisionEC_mont.py --key_type x448 --connection_data COM3
```

8.1.4.10 openssl_Ecdh_mont.py

usage: openssl_Ecdh_mont.py [-h] --key_type KEY_TYPE

[--connection_type CONNECTION_TYPE] [--connection_data CONNECTION_DATA]

Validation of Montgomery ECDH with OpenSSL engine using EC mont keys

This example showcases montgomery ECDH between openssl engine and openssl.

Precondition:

- Inject keys using openssl_provisionEC_mont.py.

optional arguments:

-h --help	show this help message and exit
--------------	---------------------------------

required arguments:

--key_type KEY_TYPE	Supported key types => x25519, x448
---------------------	-------------------------------------

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

Example invocation:

```
python openssl_Ecdh_mont.py --key_type x448
python openssl_Ecdh_mont.py --key_type x25519 --connection_type jrcpv2 --
connection_data 127.0.0.1:8050
```

8.2 Introduction on OpenSSL provider

A provider, in OpenSSL terms, is a unit of code that provides one or more implementations for various operations for diverse algorithms that one might want to perform. The key injection process is secure module specific and is not covered by the provider.

Depending on the capabilities of the attached secure element (e.g. SE050_C, SE051E, ...) the following functionality can be made available over the OpenSSL provider here (sss provider):

- EC crypto
 - EC sign/verify
 - ECDH compute key
 - EC Key Generation
 - EC CSR
- RSA crypto
 - RSA sign/verify

- RSA Key Generation
- RSA encrypt/decrypt
- RSA CSR
- Random generator

8.2.1 OpenSSL versions

The OpenSSL provider is compatible with OpenSSL versions 3.0.

8.2.2 Platforms

The OpenSSL provider can be used on iMX boards (running Linux), Raspberry Pi (running Raspbian), Windows.

Note:

When using OpenSSL provider in Linux machines, make sure OpenSSL 3.0 is installed.

8.2.3 Key formats supported

keys can be handled in two different ways in OpenSSL 3.0 provider - **Reference Key** and **Key labels**.

8.2.3.1 Reference Key

Refer to [Section 8.1.2.2EC Reference key format](#) for EC reference key format.

Refer to [Section 8.1.2.3RSA Reference key format](#) for RSA reference key format.

8.2.3.2 Key labels

KeyID's are passed with 'nxp:' prefix. Inside provider implementation we can extract the keyID from the key label and call the SSS APIs with correct reference to keys inside secure element. Example key label - nxp:0x81000000. (0x81000000 is the location of the key stored in secure element).

8.2.4 Building the OpenSSL provider

Build settings for OpenSSL provider (to be applied on top of a configured host build area):

```
cmake -DPTMW_OpenSSL:STRING=3_0 -DPTMW_HostCrypto:STRING=OPENSSL .
cmake --build .
```

The resulting OpenSSL provider will be copied to the SW tree in directory.

simw-top/sss/plugin/openssl_provider/bin.

If the OpenSSL 3.0 is not installed at default location, set the OpenSSL root directory in cmake as

```
-DOPENSSL_INSTALL_PREFIX=<OPENSSL3.0 INSTALL PATH>
```

Note:

It is mandatory to use access manager to establish PlatformSCP03 session with the secure element.

8.2.5 Sample scripts to demo OpenSSL provider

The directory `simw-top/sss/plugin/openssl_provider/scripts` contains a set of python scripts. These scripts use the OpenSSL provider in the context of standard OpenSSL utilities. They illustrate the OpenSSL provider 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 create and provision EC key material. Then it invokes the OpenSSL Provider for ECDSA sign/sign operations and ECDH calculations

```
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:

8.2.6 OSSL Algorithms property definitions

Following properties definitions are added in nxp provider,

- Random number generation - `nxp_prov.rand=yes`
- Key management - `nxp_prov.keymgmt=yes` (Required to offload the ECC / RSA keys operations to nxp provider when the keys are stored in SE05x Secure element).
 - For ECC - `nxp_prov.keymgmt.ec=yes`
 - For RSA - `nxp_prov.keymgmt.rsa=yes`
- Signature - `nxp_prov.signature=yes` (Required to offload the ECC / RSA Sign / Verify operations to nxp provider when the keys are stored in SE05x Secure element).
 - For ECDSA - `nxp_prov.signature.ecdsa=yes`
 - For RSA - `nxp_prov.signature.rsa=yes`
- Asymmetric Cipher - `nxp_prov.asym_cipher=yes` (Required to offload the RSA Encrypt / Decrypt to nxp provider when the keys are stored in SE05x Secure element).
- ECDH - `nxp_prov.keyexch=yes` (Required only when the ephemeral keys are generated on SE05x).
- Key Store - `nxp_prov.store=yes` (Required when the keys are referenced using label (nxp:) or reference keys).
 - For keys passed with nxp: prefix - `nxp_prov.store.nxp=yes`
 - For keys passed with reference key format - `nxp_prov.store.file=yes`

Note:

'fips=yes' algorithm property is added for all algorithms supported in nxp provider. This is to support the FIPS certified SE05X secure element family.

8.3 Introduction on MbedTLS ALT (2.x) 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.

8.3.1 SE05X usage in TLS handshake (ECC Cipher suites)

SE05X is used for following operations during TLS handshake

1. Verification using root CA pub key (root CA public key provisioned in SE05X)
2. Calculate shared key using `sss_derive_key_dh` (only for ECDHE cipher suites. For ECDHE, ecc key and shared secret are generated on host).

3. Sign handshake messages using provisioned client key.
4. Optional - All public key ECDSA verify operation.

8.3.2 SE05X usage in TLS handshake (RSA Cipher suites)

SE05X is used for following operations during TLS handshake

1. Verification using root CA pub key (root CA public key provisioned in SE05X).
2. Sign handshake messages using provisioned client key.
3. Optional - All public key ECDSA verify operation. Public key is set during TLS handshake.

8.3.3 Using SE05X for all Public key ECDSA verify operation

With default mbedtls config file, SE05X is used only for Root CA public key verify operation. To use secure element for all public key ecdsa verify operation, enable `MBEDTLS_ECDSSA_VERIFY_ALT` in mbedtls config file.

This feature will add some limitations as explained below.

1. NVM writes will be observed when public key is set in secured element during TLS handshake.
2. NVM writes can be avoided by overwriting the existing keys without deleting the last created key. But this limits the number of key types that can be used for handshake due to limited transient memory.

To avoid the NVM writes, modify the function `mbedtls_ecdsa_verify` in `sss/plugin/mbedtls/ecdsa_verify_alt.c`. Create the required key type object in your application and use this key object in function `mbedtls_ecdsa_verify` for verify operation. Do not delete the keyobject at the end of verify operation.

Also when `MBEDTLS_ECDSSA_VERIFY_ALT` is enabled, set the sss key store from application using api `sss_mbedtls_set_sss_keystore`. Refer example `sss/ex/mbedtls/ex_sss_ssl12.c`.

```
sss_mbedtls_set_sss_keystore(&pCtx->ks);
```

8.3.4 Using mbedtls ALT

For reference, let's look at the `sss/ex/mbedtls/ex_sss_ssl12.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;
} %%CRLF
SP% 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;
} %%CRLF
SP% /*
 * All ecdsa verification is done using the esdsa_alt file. No need to
associate the root ca pub key.
*/
#if !defined(MBEDTLS_ECDSSA_VERIFY_ALT)
    /* pex_sss_demo_tls_ctx->pub_obj will have the root CA public key */
    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;
    }%%CRLF
SP% 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;
    }
#endif%%CRLF
SP% /* 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 root CA public key from the 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 device private 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 private key from the SE for ECDH handshake.

```
sss_mbedtls_associate_ecdhctx(ssl.handshake, &pex_sss_demo_tls_ctx->obj, &pCtx-
>host_ks);
```

8.3.5 Testing on x86 / Linux

8.3.5.1 Building mbedTLS SSL/DTLS server for testing

Build mbedTLS server - :

CMake configurations:

- RTOS_Default: ON
- SSS_HAVE_HOSTCRYPTOMBEDTLS: ON
- SSS_HAVEMBEDTLS_ALT_SSS: ON
- Project: mbedtls_ex_orig_ssl_server2 / mbedtls_ex_orig_dtls_server

8.3.5.2 Building mbedTLS SSL/DTLS client (with SSS-APIs integration) for x86 / Linux

Build mbedTLS client - :

CMake configurations:

- RTOS_Default: ON
- SSS_HAVE_HOSTCRYPTOMBEDTLS: ON
- SSS_HAVEMBEDTLS_ALT_SSS: ON
- Project: mbedtls_ex_sss_ssl2_client / mbedtls_ex_sss_dtls_client

8.3.5.3 Running examples

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 9.3 Steps needed before running ssscli tool](#)
2. Provision secure element using python scripts in directory simw-top\sss\plugin\mbedtls\scripts.
Run the following commands in virtual environment:

To provision secure element for ECC

```
python3 create_and_provision_ecc_keys.py <keyType> <connection_type> <connection_string> <iot_se (optional. Default - se050)> <auth (optional. Default - None)> <auth_key>
```

To configure secure element for RSA

```
python3 create_and_provision_rsa_keys.py <keyType> <connection_type> <connection_string> <auth (optional. Default - None)> <auth_key>
```

To see possible values of input arguments, run without any parameters

```
create_and_provision_ecc_keys.py. or create_and_provision_rsa_keys.py
```

Note:

Once provisioning is done the virtual environment is not needed anymore.

3. Starting mbedTLS SSL client and server applications:

```
python3 start_ssl2_server.py <ec_curve>/<rsa_type>  
python3 start_ssl2_client.py <ec_curve>/<rsa_type> <cipher suite>  
<connection_string>
```

4. Starting mbedTLS DTLS client and server applications:

```
python3 start_dtls_server.py <ec_curve>/<rsa_type>
```

```
python3 start_dtls_client.py <ec_curve>/<rsa_type> <cipher suite>
<connection_string>
```

8.3.6 Testing mbedTLS client on frdm-k64

Run SSL Server example on Windows / Linux. Refer previous section to build server example.

8.3.6.1 Building mbedTLS SSL client (with SSS-APIs integration) for k64

The client example when run on micro-controller, takes the keys and certificates from `simw-top\sss\ex\mbedtls\mbedtls_lwip_client\ex_sss_ssl2.h`. Ensure that the file is updated with server ip address, key type and keys.

Build mbedTLS client for k64f:

CMake configurations:

- RTOS_FreeRTOS: ON
- SSS_HAVE_HOSTCRYPTOMBEDTLS: ON
- SSS_HAVEMBEDTLS_ALT_SSS: ON
- SSS_HAVE_HOST_FRDMK64F: ON
- Project: mbedtls_ex_se05x_client_lwip

8.3.6.2 Running examples

1. On x86 / Linux machine - Start mbedtls server (add `server_addr` argument) as

```
./mbedtls_ex_orig_ssl_server2 exchanges=1 force_version=tls12 debug_level=1
ca_file=simw-top\sss\plugin\mbedtls\keys\prime256v1\tls_rootca.cer
auth_mode=required key_file=simw-top\sss\plugin\mbedtls\keys
\prime256v1\tls_server_key.pem crt_file=simw-top\sss\plugin\mbedtls\keys
\prime256v1\tls_server.cer server_addr=<ip_address>
```

2. Flash mbedtls client (`mbedtls_ex_se05x_client_lwip`) on K64F,

8.3.7 SE050 Performance Measurements

The following measurements are performed on K-64 board with SE050 connected via T10I2C.

8.3.7.1 TLS1.2 using ECC Nist256 Keys (Using MbedTLS Alt)

Ciphersuite used for TLS - TLS-ECDH-ECDSA-WITH-AES-128-CBC-SHA

Read the numbers as MIN - AVG - MAX milliseconds.

Operation	SE05X (Auth - None)	K64 (with O0)	K64 (with O2)
Server Certificate Verification	49 - 49.6 - 50	1885 - 1885.4 - 1887	754 - 754.2 - 755
DH key generation	54 - 54 - 54	911 - 913.8 - 915	363 - 364.4 - 366
Sign Operation	50 - 51.8 - 59	952 - 956 - 959	382 - 384 - 386

`sss_derive_key_dh` is used for DH calculation. Time measured includes - set other party public key on host, Derive key, Get DH key from host.

8.3.7.2 TLS1.2 using RSA2048 (CRT) Keys (Using MbedTLS Alt)

Ciphersuite used for TLS - TLS-ECDHE-RSA-WITH-CHACHA20-POLY1305-SHA256

Read the numbers as MAX - AVG - MIN milliseconds.

Operation	SE05X (Auth - None)	K64 (with O0)	K64 (with O2)
Server Certificate Verification	49 - 49.8 - 50	172 - 172.2 - 172	26 - 26.8 - 27
DH key generation	NA - NA - NA	3151 - 3157.4 - 3179	813 - 834.6 - 847
Sign Operation	102 - 102.4 - 103	8450 - 8521 - 8571	1143 - 1152 - 1164

Secp521r1 key is used for DH.

8.3.8 mbedTLS ALT APIs

groupax_mbed_tls

mbedTLS ALT implementation.

Description

Implementation of key association between NXP Secure Element and mbedtls.

History

1.0 30-jan-2018 : Initial version

Functions

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 indek.

Parameters

- **handshake** – **[inout]** Pointer to the mbedtls_ssl_handshake_params which will be associated with data corresponding to the key index
- **pkeyObject** – **[in]** The object that we are going to be use.
- **hostKs** – **[in]** Keystore to host for session key.

Returns

0 if successful, or 1 if unsuccessful

int [sss_mbedtls_associate_keypair](#)(mbedtls_pk_context *pkey, [sss_object_t](#) *pkeyObject)

Associate a keypair provisioned in the secure element for subsequent operations.

Parameters

- **pkey** – **[out]** Pointer to the mbedtls_pk_context which will be associated with data corresponding to the key_index
- **pkeyObject** – **[in]** The object that we are going to be use.

Returns

0 if successful, or 1 if unsuccessful

int [sss_mbedtls_associate_pubkey](#)(mbedtls_pk_context *pkey, [sss_object_t](#) *pkeyObject)

Associate a pubkey provisioned in the secure element for subsequent operations.

Parameters

- **pkey** – **[out]** Pointer to the mbedtls_pk_context which will be associated with data corresponding to the key index
- **pkeyObject** – **[in]** The object that we are going to be use.

Returns

0 if successful, or 1 if unsuccessful

8.4 Introduction on Mbed TLS (3.x) ALT Implementation

Mbed TLS ALT implementation allows Mbed TLS 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.

Crypto operations supported -

1. ECDSA Sign. ([sss/plugin/mbedtls3x/ecdsa_sign_alt.c](#))
2. ECDSA Verify. ([sss/plugin/mbedtls3x/ecdsa_verify_alt.c](#))
3. RSA Sign. ([sss/plugin/mbedtls3x/rsa_alt.c](#))
4. RSA Verify. ([sss/plugin/mbedtls3x/rsa_alt.c](#))
5. Random number generation ([sss/plugin/mbedtls3x/ctr_drbg_alt.c](#))
6. ECDH ([sss/plugin/mbedtls3x/ecdh_alt.c](#))

Note:

In the default implementation, every time the control goes to ALT implementation, session open and close is performed. This will have all transient objects will be lost. To avoid the session open / close in ALT implementation, Use the `sss_mbedtls_set_keystore_ecdsa_sign()` / `sss_mbedtls_set_keystore_ecdsa_verify()` / `sss_mbedtls_set_keystore_rsa()` / `sss_mbedtls_set_keystore_rng()` / `sss_mbedtls_set_keystore_ecdh()` APIs to pass the keystore.

8.4.1 Key Management

Mbed TLS 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 Mbed TLS 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 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.

When the control comes to the ALT implementation, we check if the key is a reference key or not. In case of reference key, the key id is extracted and Secure element is used to perform Sign operation. If the key is not a reference key, execution will roll back to software implementation.

Refer [EC Reference key format](#) and [RSA Reference key format](#) for reference key details.

8.4.2 Build / Configuration of Mbed TLS ALT files

Mbed TLS library can be built with above ALT files using the below cmake options

CMake configurations (To be applied on top of a configured host build area):

```
cmake -DPTMW_HostCrypto:STRING=MBEDTLS -DPTMW_MBedTLS:STRING=3_X -  
DPTMW_mbedtls_ALT:STRING=SSS .
```

The custom config file to enable required ALT operation is present in `sss/plugin/mbedtls3x/mbedtls_sss_alt_config.h`. By default ECDSA Sign and RSA Sign are enabled.

Use the below macros in the config file (`sss/plugin/mbedtls3x/mbedtls_sss_alt_config.h`) to enable / disable required crypto operation

- `MBEDTLS_ECDSA_SIGN_ALT`
- `MBEDTLS_ECDSA_VERIFY_ALT`
- `MBEDTLS_RSA_ALT`
- `MBEDTLS_ECDH_COMPUTE_SHARED_ALT`

Note:

1. RSA Verify is disabled by default. Enable `SSS_ENABLE_RSA_ALT_VERIFY` in `sss/plugin/mbedtls3x/rsa_alt.c`. (Key id used for storing the public key is - 0x7D000021)

2. When ECDSA Verify / RSA Verify is enabled, KeyId - 0x7D000011(for ECC) / 0x7D000021(for RSA) is used to store the public key (a transient key is created and will reduce the available RAM memory on secure element) and do the Verify operation. Assuming first time a NIST P256 key is set, only a NIST P256 key can be overwritten here for next ECDSA Verify. Any other key type if used, will result in error during set key.

Solution - Enable `SSS_DELETE_SE05X_TMP_ECC_PUBLIC_KEY` / `SSS_DELETE_SE05X_TMP_RSA_PUBLIC_KEY` to delete the key after every Verify operation. This will ensure a new key is created next time and the implementation will work for all key types. NOTE THAT, THIS WILL RESULT IN NVM FLASH WRITES.

3. Random number generation has no ALT define in Mbed TLS config file. The `ctr_drbg.c` file is reimplemented to use Secure Element is used while building the Mbed TLS with SSS_ALT config. To enable / disable the random number generation from SE, use `SSS_ENABLE_RND_ALT` in `sss/plugin/mbedtls3x/ctr_drbg_alt.c`.

4. When using any authenticated session, make sure to disable the random number generation ALT functionality (`SSS_ENABLE_RND_ALT`), to avoid using Secure Element before the session is open.

8.4.3 SE05X usage in TLS handshake

SE05X is used for following operations during TLS handshake.

1. Sign handshake messages using provisioned client key.
2. Optional - All public key verify operation. Public key is injected / deleted during TLS handshake.

8.4.4 Testing Mbed TLS ALT (Windows)

Mbed TLS client and server example can be used to test the Mbed TLS ALT implementation.

- Project: `mbedtls_3x_client` and `mbedtls_3x_server`

Build client example with ALT option enabled (-DPTMW_mbedtls_ALT:STRING=SSS) and server example with ALT option disabled (-DPTMW_mbedtls_ALT:STRING=None)

Running examples -

Directory `simw-top\sss\plugin\mbedtls3x\scripts` contains test scripts for starting Mbed TLS server and client applications with different cipher suites. Before executing some test scripts, the secure element must first be provisioned.

1. Complete [Section 9.3 Steps needed before running ssscli tool](#)
2. Provision secure element using python scripts in directory `simw-top\sss\plugin\mbedtls3x\scripts`. Run the following commands in virtual environment:

To provision secure element for ECC

```
python3 create_and_provision_ecc_keys.py <keyType> <connection_type> <connection_string> <iot_se (optional. Default - se050)> <auth (optional. Default - None)> <auth_key>
```

To configure secure element for RSA

```
python3 create_and_provision_rsa_keys.py <keyType> <connection_type> <connection_string> <auth (optional. Default - None)> <auth_key>
```

To see possible values of input arguments, run without any parameters

```
create_and_provision_ecc_keys.py. or create_and_provision_rsa_keys.py
```

Note:

Once provisioning is done the virtual environment is not needed anymore. when running with vcom export port using EX_SSS_BOOT_SSS_PORT=<COM_PORT>.

3. Starting Mbed TLS client and server applications:

```
python3 start_ssl2_server.py <ec_curve>/<rsa_type>  
python3 start_ssl2_client.py <ec_curve>/<rsa_type> <cipher suite>
```

8.5 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](#).

8.5.1 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.

8.5.2 PSA Concepts

Location

The location of an SE driver is the unique identifier it is registered with. To access any functionality this value is checked by the PSA stack.

Lifetime

The lifetime of an object refers to its persistence. An object can either be PERSISTENT or VOLATILE. Lifetime identifies the scope of an object (where the object is stored and which operations are available for it). For SE driver objects, this value is defined as `(PSA_ALT_SE05X_LOCATION << 8) + PERSISTENCE_VALUE` where `PERSISTENCE_VALUE` indicates the `psa_key_persistence_t` value of the object (volatile/persistent). 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.

Also see [Managing KeyIDs](#).

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 meta data 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 may also stored on SE.

Also see [Managing KeyIDs](#).

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. This is also used to continue a pending operation if, in case, the system reboots.
- **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. Also see [Managing KeyIDs](#).

Warning:

This logic of managing KeyIDs of various PSA objects is temporary and it will be changed in the future.

8.5.3 Managing KeyIDs

Application can provide any keyID to be used with SE. This will be mapped directly (1:1) with the slotID. For internal usage, it is mapped to a 28-bit ID, masking out the most significant nibble. Of all PSA objects, **lifetime file**, **transaction file** and **secure objects** will always be stored in secure element. The application has an option to choose the storage for object meta data files.

Providing the mask `#define PSA_ALT_ITS_SE_FLAG ((0x1) << 28)` in the keyID will ensure that object file is stored in secure element. If this flag is not set, object file will be stored in flash memory.

Flash storage currently has support to store only up to 8 object files.

8.5.4 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=lpcxpresso55s_s
- HostCrypto=MBEDTLS
- mbedTLS_ALT=PSA
- RTOS=Default
- SMCOM=T1oI2C
- PROJECT=PSA_ALT

Note:

In case an example using this library freezes after registering the driver,

try again after deleting the flash memory. This is needed as the PSA keystore is persistent and does not get deleted during flashing the program. If this does not work, pass true to the function call psa_flash_ks_init() in file psa_secure.c. This will erase the previous keystore and will create a new persistent keystore. After this, the original program will run without needing this change.

8.6 Android Key master

See [Android: Hikey960](#)

8.7 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

8.7.1 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\security Policies`), 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](#))

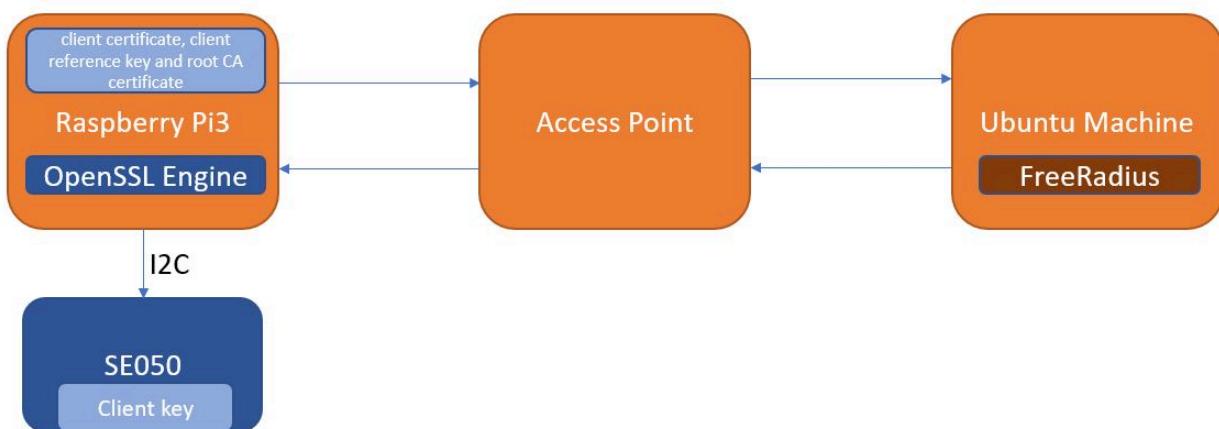
8.8 WiFi EAP Demo with Raspberry Pi3

8.8.1 Prerequisites

- Raspberry pi3 with Raspbian OS installed.
- Ubuntu machine to run freeRadius
- Access point (WPA/WPA2 Enterprise capable)

8.8.2 Introduction

The image shows the wifi EAP demo set up



8.8.3 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

8.8.4 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/
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 freeradiux -X
```

8.8.5 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_t1oi2c
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 ssscli tool setup. Using ssscli tool, create a reference pem file for client key

```
cd /home/pi/wifiEAP
openssl rsa -in client.key -out client.pem
ssscli connect se05x t1oi2c none
ssscli set rsa pair 0x1234 client.pem
ssscli refpem rsa pair 0x1234 client_ref.pem
```

6. Add folowing network configuration to wpa_supplicant.conf file (/etc/wpa_supplicant)

```
pkcs11_engine_path=/usr/local/lib/libssss_engine.so
pkcs11_module_path=/usr/local/lib/libssss_engine.so

network={
    ssid=""
    priority=1
    engine=1
    key_mgmt=WPA-EAP
    pairwise=CCMP TKIP
    auth_alg=OPEN
    eap=TTLS
    use TLS
    identity=""                      # from user configuration file
    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=""      # If key
    file is not encrypted with pass phrase, comment this line.

    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=""        # If key
    file is not encrypted with pass phrase, comment this line.
}
```

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
dynamic_path = /usr/local/lib/libssss_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.*
3. *The example is last tested with Plug and Trust Middleware version 4.5.1.*

8.9 PKCS#11 Standalone Library

PKCS#11(v2.40) 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.

8.9.1 PKCS#11 Label Handling

PKCS#11 library has three ways to calculate keyId through `LabelToKeyId()`:

1. **If `labelSize == 0`**
 - keyId is generated through a random generator.
2. **If `CKA_LABEL` starts with `sss`:**
 - keyID is generated by interpreting following string as hex value of the keyID. Example - If `CKA_LABEL` is `sss:20181001`, keyID is `0x20181001`
3. **Any other `CKA_LABEL`**
 - KeyId is generated as the last 4 bytes of SHA512 digest of label.

8.9.2 PKCS#11 Object caching

PKCS#11 library supports caching objects during `C_FindObjects()` operation which can improve execution time of subsequent functions. The objects are cached during `C_FindObjects()` operation and are valid until `C_FindObjectsFinal()` is called. On calling `C_FindObjectsFinal()`, the cached objects are invalidated.

The caching feature is enabled by default for MPUs and disabled by default for MCUs. (Macro - `ENABLE_OBJECT_CACHE` in file `sss_pkcs11_pal.h`).

8.9.3 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

Supported API

- General-purpose functions
 - C_Initialize
 - C_Finalize
 - C_GetInfo
 - C_GetFunctionList
- Slot and token management functions
 - C_GetSlotList
 - C_GetSlotInfo
 - C_GetTokenInfo
 - C_GetMechanismList
 - C_GetMechanismInfo
- Session management functions
 - C_OpenSession
 - C_CloseSession
 - C_GetSessionInfo
 - C_Login

- C_Logout
- Object management functions
 - C_CreateObject
 - C_DestroyObject
 - C_GetAttributeValue
 - C_FindObjectsInit
 - C_FindObjects
 - C_FindObjectsFinal
- Encryption functions
 - C_EncryptInit
 - C_Encrypt
- Decryption functions
 - C_DecryptInit
 - C_Decrypt
- Message digesting functions
 - C_DigestInit
 - C_Digest
 - C_DigestUpdate
 - C_DigestFinal
- Signing and MACing functions
 - C_SignInit
 - C_Sign
 - C_VerifyInit
 - C_Verify
- Key management functions
 - C_GenerateKey
 - C_GenerateKeyPair
 - C_DeriveKey
- Random number generation functions
 - C_SeedRandom
 - C_GenerateRandom

Note:

The multi-step operations of ECDSA Sign/Verify, RSA Sign/Verify, HMAC, all will use the host crypto for multi-step digest operations of large input data.

8.9.4 Building on Linux / Raspberry-Pi

PKCS#11 standalone shared library can be built on Linux platforms and Raspberry-Pi. Build project: `sss_pkcs11` for PKCS#11 library.

Note:

While using PKCS#11 as a library on multithreaded systems, the application must ensure proper locking is used. Calling multiple APIs from the library from different threads without proper locks can lead to unexpected behaviour.

8.9.5 Using with pkcs11-tool

Install `pkcs11-tool` by running:

```
sudo apt-get install opensc-pkcs11
```

Set environment variable to the installed PKCS#11 shared library:

```
export PKCS11_MODULE=/usr/local/lib/libsss_pkcs11.so
```

The `.so` file is available in `binaries/pkcs11` directory.

Generating new keypair:

```
pkcs11-tool --module $PKCS11_MODULE --keypairgen --key-type rsa:1024 --label "sss:202020"
```

Sigining:

```
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
```

8.9.6 Notes

The monotonic counter will increase by one each time its value is read as specified in “PKCS #11 Cryptographic Token Interface Base Specification Version 2.40”. This will cause NVM write accesses.

9 CLI Tool

9.1 Introduction

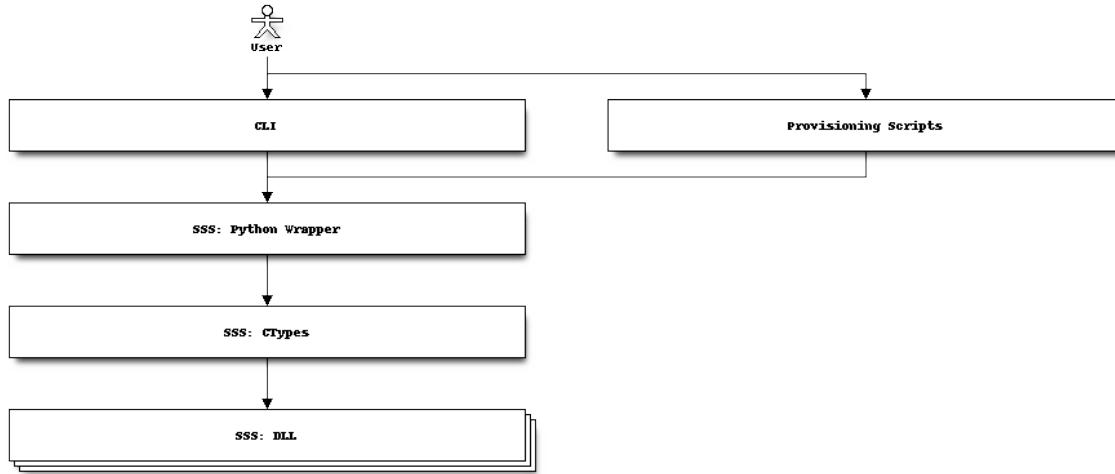
The `sscli` tool is a Python based tool to manipulate the SE050 Secure Element.

It can be used to:

- Insert Keys, Certificates
- Generate Keys
- Attach policies to objects
- 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.

9.2 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 Plug & Trust MW : Block Diagram . Based on the selection of the platform/interface, this DLL needs to be re-compiled.

9.3 Steps needed before running `ssscli` tool

9.3.1 Once per installation

The following steps are needed once per installation.

9.3.1.1 Windows

1. Ensure 32 bit Python 3.8 or later 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
```

- Once virtualenv is installed, create a new virtual environment:

```
python -m virtualenv venv
```

- To activate the new created virtual ENV Run:

```
call venv\Scripts\activate.bat
```

- In the new installed virtualenv, install required packages. This includes click and cryptography

- click: To parse command line parameter.
- cryptography: Load keys, certificates and generate reference keys.

Change directory to <SE05X_root_folder>/simw-top/pycli and run:

```
pip install -r requirements.txt
```

- To install ssscli tool, run the following commands:

```
cd src  
pip install --editable .
```

- Alternately can install ssscli tool, by running following commands:

```
cd src  
python setup.py develop
```

9.3.1.2 i.MX

- Ensure Python 3.8 or later is installed.
- Ensure func-timeout module is installed.
- Follow [Section 4.4i.MX Linux Build](#) and build the sssapisw.
- To install ssscli tool, change directory to <SE05X_root_folder>/simw-top/pycli/src and run the following command:

```
python3 setup.py develop
```

9.3.1.3 Raspberry Pi

- Ensure Python 3.8 or later is installed .
- Ensure python3-pip, python3-dev and libffi-dev are installed:

```
sudo apt-get install python3-pip python3-dev libffi-dev
```

- Follow [Section 4.5Raspberry Pi Build](#) and build the sssapisw.

Run:

```
sudo apt install python3-virtualenv
```

- Once virtualenv is installed, create a new virtual environment:
virtualenv venv

- To activate the new created virtual ENV Run:

```
source venv/bin/activate.bat
```

6. Install click, cryptography and func-timeout modules in virtual environment. To install these modules, change directory to <SE05X_root_folder>/simw-top/pycli and run the following command:
pip install -r requirements.txt
7. To install ssscli tool, run the following command in virtual environment:
cd src pip install --editable .

9.3.2 Communication interface (cmake SMCOM setting)

The communication interface to the secure element used by the ssscli tool is determined by the native shared library used. The communication interface supported by the shared library is controlled by the cmake SMCOM value.

To change the communication interface only the native ssscli DLL / .so needs to be rebuilt.

The shared library can be compiled with MSVC/MinGW/XCode/GCC based on the selected platform.

9.4 Running the ssscli tool - Windows

1. To activate the new created virtual ENV Run:

```
call venv\Scripts\activate.bat
```

2. To connect to the SE, call:

```
ssscli connect se05x vcom COM13
```

3. To dis-connect to the SE, call:

```
ssscli disconnect
```

9.5 CLI Provisioning

9.5.1 Generating keys and certificates

For generating keys and certificates, following scripts generates using openssl.

- GenerateAWS Credentials.py
- GenerateAZURE Credentials.py

The generated keys and certificates shall be available in pycli/Provisioning/azure and pycli/Provisioning/aws directories.

9.5.2 Provisioning for the demo

Generated keys and certificates are used to provision the secure element using ResetAndUpdate_AZURE.py and ResetAndUpdate_AWS.py scripts for azure and aws cloud demo respectively.

Note:

Default auth type in provisioning script is always None

9.5.3 Steps to provision your device for demo on Windows

Provisioning on windows can be done in two ways.

- Using precompiled binaries
- Using python scripts

9.5.3.1 Using precompiled binaries

Precompiled binaries available in `binaries/PCWindows/ssscli` directory. Can generate certificates and provision the secure element by simply running these binaries.

1. For AWS, create certificate and provision, call:

```
Provision_AWS.exe <COM_PORT>
```

2. For AZURE, create certificate and provision, call:

```
Provision_AZURE.exe <COM_PORT>
```

The generated keys and certificates shall be available in `binaries/PCWindows/ssscli/aws` and `binaries/PCWindows/ssscli/azure` directories.

9.5.3.2 Using Python scripts

1. Complete [Section 9.3 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 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>
```

5. 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>
```

6. Flash the demo on to the board

9.5.4 Steps to provision your device for demo on iMX or Raspberry Pi

1. Complete [Section 9.3 Steps needed before running ssscli tool](#)

2. from `pycli` directory, run:

```
cd Provisioning
```

3. For AWS, create certificate and provision, call:

```
python3 GenerateAWSCredentials.py  
python3 verification_certificate.py <interCA_Certificate> <interCA_Keypair>  
    <verification_code>  
python3 ResetAndUpdate_AWS.py
```

4. For AZURE, create certificate and provision, call:

```
python3 GenerateAZURECredentials.py  
python3 verification_certificate.py <interCA_Certificate> <interCA_Keypair>  
    <verification_code>  
python3 ResetAndUpdate_AZURE.py
```

5. Flash the demo on to the board

9.6 Usage Examples

9.6.1 SE05X: VCOM Interface

Provisioning ECC Pair and Certificate:

```
ssscli connect se05x 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 retrive public key:

```
ssscli connect se05x 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 se05x 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 se05x 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 se05x 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 se05x 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 se05x 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 Binary data and verify:

```
ssscli connect se05x vcom COM5
ssscli generate ecc 12E41001 Secp256k1
ssscli sign 12E41001 tstData\binary_data.bin signed_data_ecc_secp256k1.pem
ssscli verify 12E41001 tstData\binary_data.bin signed_data_ecc_secp256k1.pem
ssscli disconnect
```

Generate ecc Brainpool192 Key and Sign and verify certificate using SHA512 has algorithm:

```
ssscli connect se05x vcom COM5
ssscli generate ecc 0xE101501 Brainpool192
ssscli sign 0xE101501 tstData\tls_client.cer data\signed_data.pem --
hashalgo=SHA512
ssscli verify 0xE101501 tstData\tls_client.cer data\signed_data.pem --
hashalgo=SHA512
ssscli disconnect
```

Read Cert UID of 10 bytes long:

```
ssscli connect se05x vcom COM5
ssscli se05x certuid
```

Read UID of 18 bytes long:

```
ssscli connect se05x vcom COM5
ssscli se05x uid
```

Session open with auth type as Platform SCP, generate ecc Brainpool192 Key:

```
ssscli connect se05x vcom COM5 --auth_type PlatformSCP --scpkey "c:/_ddm/
scpkey.txt"
ssscli se05x reset
ssscli generate ecc 2E10D532 Brainpool192
ssscli disconnect
```

Generate ecc Koblitz256 key and create reference key:

```
ssscli connect se05x 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 se05x 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 se05x 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 se05x vcom COM5  
ssscli generate ecc 70102060 ED_25519  
ssscli sign 70102060 testData\tls_client.cer data\signed_data_using_ed25519.pem  
ssscli disconnect
```

Generate ecc MONTH DH 25519 key:

```
ssscli connect se05x vcom COM5  
ssscli generate ecc 70102080 ED_25519  
ssscli sign 70102080 testData\tls_client.cer data\signed_data_using_ed25519.pem  
ssscli disconnect
```

Perform Encrypt and Decrypt using RSA 2048:

```
ssscli connect se05x 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
```

Provision and Retrieve Binary data:

```
ssscli connect se05x vcom COM5  
ssscli set bin 0x20191005 testData\binary_data.hex  
ssscli get bin 0x20191005 binary_data.hex  
ssscli disconnect
```

9.6.2 SE05X: PCSC interface

Provisioning ECC Pair and Certificate:

```
ssscli connect se05x pcsc NXP  
ssscli se05x reset  
ssscli set ecc pair 0x20181001 testData\tls_client_key.pem  
ssscli set cert 0x20181002 testData\tls_client.cer  
ssscli disconnect
```

Inject ECC Public Key:

```
ssscli connect se05x pcsc NXP  
ssscli set ecc pub 0x20181010 testData\tls_client_key_pub.pem  
ssscli disconnect
```

9.6.3 se05x: JRCPV2 interface

Provisioning ECC Pair and Certificate:

```
ssscli connect se05x jrcpv2 127.0.0.1:8050
ssscli se05x reset
ssscli set ecc pair 0x20181001 testData\tls_client_key.pem
ssscli set cert 0x20181002 testData\tls_client.cer
ssscli disconnect
```

Generating ecc key and retrive public key:

```
ssscli connect se05x jrcpv2 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 se05x jrcpv2 127.0.0.1:8050
ssscli set cert 0x20181002 testData\tls_client.cer
ssscli get cert 0x20181002 data\extracted_certificate.cer
ssscli disconnect
```

Erase a key, Inject ecc Key and Sign certificate:

```
ssscli connect se05x jrcpv2 127.0.0.1:8050
ssscli erase 0x20181001
ssscli set ecc pair 0x20181001 testData\tls_client_key.pem
ssscli sign 0x20181001 testData\tls_client.cer data\signed_data.pem
ssscli disconnect
```

Inject and Retrieve AES key:

```
ssscli connect se05x jrcpv2 127.0.0.1:8050
ssscli se05x reset
ssscli set aes 0x40200000 testData\aes.der
ssscli get aes 0x40200000 data\extracted_aes_key.cer
ssscli disconnect
```

Inject ECC Public Key:

```
ssscli connect se05x jrcpv2 127.0.0.1:8050
ssscli set ecc pub 0x20181010 testData\tls_client_key_pub.pem
ssscli disconnect
```

Generate RSA Key and Get public key in PEM format:

```
ssscli connect se05x 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 Binary data and verify:

```
ssscli connect se05x jrcpv2 127.0.0.1:8050
```

```
ssscli generate ecc 12E41001 Secp256k1
ssscli sign 12E41001 tstData\binary_data.bin signed_data_ecc_secp256k1.pem
ssscli verify 12E41001 tstData\binary_data.bin signed_data_ecc_secp256k1.pem
ssscli disconnect
```

Generate ecc Brainpool192 Key and Sign and verify certificate using SHA512 has algorithm:

```
ssscli connect se05x 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 se05x jrcpv2 127.0.0.1:8050
ssscli se05x certuid
```

Read UID of 18 bytes long:

```
ssscli connect se05x jrcpv2 127.0.0.1:8050
ssscli se05x uid
```

Session open with auth type as Platform SCP, generate ecc Brainpool192 Key:

```
ssscli connect se05x jrcpv2 127.0.0.1:8050 --auth_type PlatformSCP --scpkey
"c:/ddm/scpkey.txt"
ssscli se05x reset
ssscli generate ecc 2E10D532 Brainpool192
ssscli disconnect
```

Generate ecc Koblitz256 key and create reference key:

```
ssscli connect se05x 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 se05x 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 se05x jrcpv2 127.0.0.1:8050
ssscli generate ecc 70102050 Brainpool256
ssscli refpem ecc pair 70102050 ecc_bp256_rekey.pem --format PKCS12 --password
npx
ssscli disconnect
```

Generate ecc ED25519 key and sign certificate:

```
ssscli connect se05x jrcpv2 127.0.0.1:8050
ssscli generate ecc 70102060 ED_25519
ssscli sign 70102060 tstData\tls_client.cer data\signed_data_using_ed25519.pem
ssscli disconnect
```

Generate ecc MONTH DH 25519 key:

```
ssscli connect se05x jrcpv2 127.0.0.1:8050
ssscli generate ecc 70102080 ED_25519
ssscli sign 70102080 tstData\tls_client.cer data\signed_data_using_ed25519.pem
ssscli disconnect
```

Perform Encrypt and Decrypt using RSA 2048:

```
ssscli connect se05x 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
```

Provision and Retrieve Binary data:

```
ssscli connect se05x jrcpv2 127.0.0.1:8050
ssscli set bin 0x20191005 tstData\binary_data.hex
ssscli get bin 0x20191005 binary_data.hex
ssscli disconnect
```

Get available Free memory

```
ssscli connect se05x jrcpv2 127.0.0.1:8050
ssscli se05x getmemory
ssscli disconnect
```

9.6.4MBEDTLS

Portname is to be passed as a existing directory for keystore.

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 retrive 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
```

9.7 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.

9.7.1 ssscli Commands

These are the top level commands accepted by the ssscli Tool.

1. ssscli:

```
Usage: ssscli [OPTIONS] COMMAND [ARGS]...%%CRLF
SP%% Command line interface for SE050

Options:
  -v, --verbose    Enables verbose mode.
  --version        Show the version and exit.
  --help           Show this message and exit.

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
  policy         Create/Dump Object Policy
  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%%CRLF
SP%% Open Session.%%CRLF
```

```
SP%% subsystem = Security subsystem is selected to be used. Can be one of
"se05x,
auth, mbedtls, openssl"%%CRLF
SP%% method = Connection method to the system. Can be one of "none, sci2c,
vcom,
t1oi2c, jrcpv1, jrcpv2, pcsc"%%CRLF
SP%% port_name = Subsystem specific connection parameters. Example: COM6,
127.0.0.1:8050. Use "None" where not applicable. e.g. SCI2C/T1oI2C. Default
i2c port (i2c-1) will be used for port name = "None".
```

Options:

```
--auth_type [None|PlatformSCP|UserID|ECKey|AESKey|UserID_PlatformSCP|
ECKey_PlatformSCP|AESKey_PlatformSCP]
--scpkey TEXT
--help
```

Authentication type. Default is "None". Can be one of "None, UserID, ECKey, AESKey, PlatformSCP, UserID_PlatformSCP, ECKey_PlatformSCP, AESKey_PlatformSCP"

File path of the platformscp keys for platformscp session

Show this message and exit.

3. ssscli disconnect:

```
Usage: ssscli disconnect [OPTIONS] %%CRLF
SP%% Close session.
```

Options:

```
--help Show this message and exit.
```

4. ssscli set:

```
Usage: ssscli set [OPTIONS] COMMAND [ARGS]... %%CRLF
SP%% Set ECC/RSA/AES Keys or certificates
```

Options:

```
--help Show this message and exit.
```

Commands:

```
aes    Set AES Keys
bin    Set Binary
cert   Set Certificate
ecc    Set ECC Keys
hmac   Set HMAC Keys
rsa    Set RSA Keys
```

5. ssscli get:

```
Usage: ssscli get [OPTIONS] COMMAND [ARGS]... %%CRLF
SP%% Get ECC/RSA/AES Keys or certificates
```

Options:

```
--help Show this message and exit.
```

Commands:

```
aes    Get AES Keys
bin    Get Binary
cert   Get Certificate
ecc    Get ECC Keys
rsa    Get RSA Keys
```

6. ssscli generate:

```
Usage: ssscli generate [OPTIONS] COMMAND [ARGS]...%%CRLF
SP%% 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%%CRLF
SP%% Erase ECC/RSA/AES Keys or Certificate (contents)%%CRLF
SP%% keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001
```

Options:

```
--help Show this message and exit.
```

8. ssscli cloud:

```
Usage: ssscli cloud [OPTIONS] COMMAND [ARGS]...%%CRLF
SP%% (Not Implemented) Cloud Specific utilities.%%CRLF
SP%% 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
gcp  (Not Implemented) GCP (Google Cloud Platform) Specific utilities
ibm  (Not Implemented) IBM Watson Specific utilities
```

9. ssscli se05x:

```
Usage: ssscli se05x [OPTIONS] COMMAND [ARGS]...%%CRLF
SP%% SE05X specific commands
```

Options:

```
--help Show this message and exit.
```

Commands:

```
certuid      Get SE05X Cert Unique ID (10 bytes)
getmemory    Get available memory
getrng       Get SE05X random number (10 bytes)
readidlist   Read contents of SE050
reset        Reset SE05X
uid          Get SE05X Unique ID (18 bytes)
```

10. ssscli refpem:

```
Usage: ssscli refpem [OPTIONS] COMMAND [ARGS]...%%CRLF
SP%% Create Reference PEM/DER files (For OpenSSL Engine).
```

Options:

```
--help Show this message and exit.
```

Commands:

```
ecc  Refpem ECC Keys
```

```
rsa Refpem RSA Keys
```

11. ssscli sign:

```
Usage: ssscli sign [OPTIONS] keyid input_file signature_file%%CRLF
SP%% Sign Operation%%CRLF
SP%% keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001%%CRLF
SP%% input_file = Input file to sign. By default filename with extension .pem
and
.cer considered as PEM format, others as DER/BINARY format.%%CRLF
SP%% signature_file = File name to store signature data. By default filename
with
extension .pem in PEM format and others in DER format.

Options:
--informat TEXT      Input 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.
```

12. ssscli verify:

```
Usage: ssscli verify [OPTIONS] keyid input_file signature_file%%CRLF
SP%% verify operation%%CRLF
SP%% keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001%%CRLF
SP%% input_file = Input file to verify. By default filename with
extension .pem
and .cer considered as PEM format, others as DER/BINARY format.%%CRLF
SP%% filename = signature_file data file for verification. By default
filename
with extension .pem in PEM format and others in DER format.
```

```
Options:
--format TEXT      input_file and signature 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 encrypt:

```
Usage: ssscli encrypt [OPTIONS] keyid input_data filename%%CRLF
SP%% Sign Operation%%CRLF
SP%% keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001%%CRLF
SP%% input_data = Input data to Encrypt. can be raw string or in file.%%CRLF
SP%% 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.
```

14. ssscli decrypt:

```
Usage: ssscli decrypt [OPTIONS] keyid encrypted_data filename%%CRLF  
SP%% Sign Operation%%CRLF  
SP%% keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001%%CRLF  
SP%% encrypted_data = Encrypted data to Decrypt. can be raw data or in file.  
Input data should be in DER format.%%CRLF  
SP%% 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.
```

15. ssscli policy:

```
Usage: ssscli policy [OPTIONS] COMMAND [ARGS]...%%CRLF  
SP% Create/Dump Object Policy
```

```
Options:  
  --help Show this message and exit.
```

```
Commands:  
  asymkey  Create Asymmetric Key Object Policy  
  counter   Create Counter Object Policy  
  dump     Display Created Object Policy  
  file      Create Binary file Object Policy  
  pcr       Create PCR Object Policy  
  symkey   Create Symmetric Key Object Policy  
  userid    Create User ID Object Policy
```

9.7.2 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%%CRLF  
SP% Set AES Keys%%CRLF  
SP% keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001%%CRLF  
SP% key = Can be in file or raw key in DER or HEX format  
  
Options:  
  --policy_name TEXT  File name of the policy to be applied  
  --help             Show this message and exit.
```

2. ssscli set hmac:

```
Usage: ssscli set hmac [OPTIONS] keyid key%%CRLF  
SP% Set HMAC Keys%%CRLF  
SP% keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001%%CRLF  
SP% 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%%CRLF
SP%% Set Certificate%%CRLF
SP%% keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001%%CRLF
SP%% 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"
--policy_name TEXT      File name of the policy to be applied
--help                  Show this message and exit.
```

4. ssscli set ecc pair:

```
Usage: ssscli set ecc pair [OPTIONS] keyid key%%CRLF
SP%% Set ECC Key pair%%CRLF
SP%% keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001%%CRLF
SP%% 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"
--policy_name TEXT      File name of the policy to be applied
--help                  Show this message and exit.
```

5. ssscli set ecc pub:

```
Usage: ssscli set ecc pub [OPTIONS] keyid key%%CRLF
SP%% Set ECC Public Keys%%CRLF
SP%% keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001%%CRLF
SP%% 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"
--policy_name TEXT      File name of the policy to be applied
--help                  Show this message and exit.
```

6. ssscli set rsa pair:

```
Usage: ssscli set rsa pair [OPTIONS] keyid key%%CRLF
SP%% Set RSA Key Pair%%CRLF
SP%% keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001%%CRLF
SP%% 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"
--policy_name TEXT      File name of the policy to be applied
--help                  Show this message and exit.
```

7. ssscli set rsa pub:

```
Usage: ssscli set rsa pub [OPTIONS] keyid key%%CRLF
SP%% Set RSA Public Keys%%CRLF
```

```
SP%% keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001%%CRLF
SP%% 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"
--policy_name TEXT  File name of the policy to be applied
--help             Show this message and exit.
```

8. ssscli set bin:

```
Usage: ssscli set bin [OPTIONS] keyid data%%CRLF
SP%% Set Certificate%%CRLF
SP%% keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001%%CRLF
SP%% data = Can be raw binary or in file
```

Options:

```
--policy_name TEXT  File name of the policy to be applied
--help             Show this message and exit.
```

9.7.3 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%%CRLF
SP%% Get AES Keys%%CRLF
SP%% keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001%%CRLF
SP%% 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%%CRLF
SP%% Get Certificate%%CRLF
SP%% keyid = 32bit Key ID. Should be in hex format. Example: 401286E6%%CRLF
SP%% 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%%CRLF
SP%% Get ECC Pair%%CRLF
SP%% keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001%%CRLF
SP%% 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%%CRLF  
SP%% Get ECC Pub%%CRLF  
SP%% keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001%%CRLF  
SP%% 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%%CRLF  
SP%% Get RSA Pair%%CRLF  
SP%% keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001%%CRLF  
SP%% 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%%CRLF  
SP%% Get RSA Pub%%CRLF  
SP%% keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001%%CRLF  
SP%% 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. ssscli get bin:

```
Usage: ssscli get bin [OPTIONS] keyid filename%%CRLF  
SP%% Get Binary%%CRLF  
SP%% keyid = 32bit Key ID. Should be in hex format. Example: 401286E6%%CRLF  
SP%% filename = File name to store binary data.
```

Options:

```
--help Show this message and exit.
```

9.7.4 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|BN_P256}%%CRLF
SP%% Generate ECC Key%%CRLF
SP%% keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001%%CRLF
SP%% 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:
--policy_name TEXT File name of the policy to be applied
--help             Show this message and exit.
```

2. ssscli generate rsa:

```
Usage: ssscli generate rsa [OPTIONS] keyid {1024|2048|3072|4096}%%CRLF
SP%% Generate RSA Key%%CRLF
SP%% keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001%%CRLF
SP%% bits = Number of bits. can be one of "1024, 2048, 3072, 4096"

Options:
--policy_name TEXT File name of the policy to be applied
--help             Show this message and exit.
```

9.7.5 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%%CRLF
SP%% Create reference PEM file for ECC Pair%%CRLF
SP%% keyid = 32bit Key ID. Should be in hex format. Example: 0x20E8A001%%CRLF
SP%% 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.
--cert TEXT        Certificate for PKCS12 format.
--help             Show this message and exit.
```

2. ssscli refpem ecc pub:

```
Usage: ssscli refpem ecc pub [OPTIONS] keyid filename%%CRLF
SP%% Create reference PEM file for ECC Pub%%CRLF
```

```
SP%% keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001%%CRLF
SP%% 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.
--cert TEXT        Certificate for PKCS12 format.
--help             Show this message and exit.
```

3. ssscli refpem rsa pair:

```
Usage: ssscli refpem rsa pair [OPTIONS] keyid filename%%CRLF
SP%% Create reference PEM file for RSA Pair%%CRLF
SP%% keyid = 32bit Key ID. Should be in hex format. Example: 20E8A001%%CRLF
SP%% 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.
--cert TEXT        Certificate for PKCS12 format.
--help             Show this message and exit.
```

9.7.6 Se05x Commands

These are the SE05x specific commands.

1. ssscli se05x uid:

```
Usage: ssscli se05x uid [OPTIONS]%%CRLF
SP%% 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]%%CRLF
SP%% 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]%%CRLF
SP%% Resets the SE05X Secure Module to the initial state.%%CRLF
SP%% 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.%%CRLF
SP%% It does not use the low level SE05X API ``Se05x_API_DeleteAll``%%CRLF
SP%% 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]%%CRLF
SP%% Read contents of SE050

Options:
--help Show this message and exit.

5. ssscli se05x getmemory:
Usage: ssscli se05x getmemory [OPTIONS]%%CRLF
SP%% Gets the amount of free memory.

Options:
--help Show this message and exit.
```

9.7.7 POLICY Commands

These are Policy commands.

```
1. ssscli policy asymkey:
Usage: ssscli policy asymkey [OPTIONS] policy_name auth_obj_id%%CRLF
SP%% Create Asymmetric key object policy.%%CRLF
SP%% policy_name = File name of the policy to be created. This policy name
should
be given as input while provisioning.%%CRLF
SP%% auth_obj_id = Auth object id for each Object Policy.

Options:
--sign BOOLEAN Object policy Allow Sign. Enabled by
Default
--verify BOOLEAN Object policy Allow Verify. Enabled by
Default
--encrypt BOOLEAN Object policy Allow Encryption. Enabled by
Default
--decrypt BOOLEAN Object policy Allow Decryption. Enabled by
Default
--key_derive BOOLEAN Object policy Allow Key Derivation.
Disabled
--wrap BOOLEAN by Default
Object policy Allow Wrap. Disabled by
Default
--generate BOOLEAN Object policy Allow Generate. Enabled by
Default
--write BOOLEAN Object policy Allow Write. Enabled by
Default
--read BOOLEAN Object policy Allow Read. Enabled by
Default
Default
--import_export BOOLEAN Object policy Allow Import Export. Disabled
by Default
--key_agreement BOOLEAN Object policy Allow Key Agreement. Disabled
by Default
--attest BOOLEAN Object policy Allow attestation. Disabled
by
Default
--forbid_derived_output BOOLEAN Object policy Forbid Derived Output.
Disabled by Default
```

--forbid_all BOOLEAN	Object policy forbid all. Disabled by Default
--delete BOOLEAN	Object policy Allow Delete. Enabled by Default
--req_sm BOOLEAN	Object policy Require Secure Messaging. Disabled by Default
--req_pcr_val BOOLEAN by	Object policy Require PCR Value. Disabled by Default
--pcr_obj_id TEXT	Object policy PCR object ID in HEX format. Zero by Default
--pcr_expected_value TEXT	Object policy PCR Expected in Hex byte string Value. Zero by Default
--help	Show this message and exit.

2. ssscli policy symkey:

Usage: ssscli policy symkey [OPTIONS] policy_name auth_obj_id%%CRLF
 SP%% Create Symmetric key object policy.%%CRLF
 SP%% policy_name = File name of the policy to be created. This policy name should be given as input while provisioning.%%CRLF
 SP%% auth_obj_id = Auth object id for each Object Policy.

Options:

--sign BOOLEAN Default	Object policy Allow Sign. Enabled by Default
--verify BOOLEAN	Object policy Allow Verify. Enabled by Default
--encrypt BOOLEAN	Object policy Allow Encryption. Enabled by Default
--decrypt BOOLEAN	Object policy Allow Decryption. Enabled by Default
--key_derive BOOLEAN Disabled	Object policy Allow Key Derivation. by Default
--wrap BOOLEAN	Object policy Allow Wrap. Disabled by Default
--generate BOOLEAN	Object policy Allow Generate. Disabled by Default
--write BOOLEAN	Object policy Allow Write. Enabled by Default
--read BOOLEAN Default	Object policy Allow Read. Enabled by Default
--import_export BOOLEAN	Object policy Allow Import Export. Disabled by Default
--desfire_auth BOOLEAN	Object policy Allow to perform DESFire authentication. Disabled by Default
--desfire_dump BOOLEAN	Object policy Allow to dump DESFire session keys. Disabled by Default
--forbid_derived_output BOOLEAN	Object policy Forbid Derived Output. Disabled by Default
--kdf_ext_random BOOLEAN	Object policy Allow key derivation ext random. Disabled by Default
--tls_kdf BOOLEAN	Object policy Allow tls kdf. Disabled by Default
--tls_pms_kd BOOLEAN	Object policy Allow tls pms kd. Disabled by Default
--hkdf BOOLEAN Default	Object policy Allow hkdf. Enabled by Default

--pbkdf BOOLEAN	Object policy Allow pbkdf. Disabled by Default
--desfire_kd BOOLEAN	Object policy Allow desfire kd. Disabled by Default
--forbid_external_iv BOOLEAN	Object policy forbid external IV. Disabled by Default
--usage_hmac_pepper BOOLEAN	Object policy Allow usage hmac as pepper. Disabled by Default
--desfire_change_key BOOLEAN	Object policy Allow desfire change key. Disabled by Default
--derived_input BOOLEAN	Object policy Allow derived input. Disabled by Default
--desfire_auth_id TEXT	32 bit desfire auth id for desfire_change_key policy
--source_key_id TEXT	32 bit source key id for derived_input policy
--forbid_all BOOLEAN	Object policy forbid all. Disabled by Default
--delete BOOLEAN	Object policy Allow Delete. Enabled by Default
--req_sm BOOLEAN	Object policy Require Secure Messaging. Disabled by Default
--req_pcr_val BOOLEAN	Object policy Require PCR Value. Disabled by Default
--pcr_obj_id TEXT	Object policy PCR object ID in HEX format. Zero by Default
--pcr_expected_value TEXT	Object policy PCR Expected in Hex byte string Value. Zero by Default
--help	Show this message and exit.

3. ssscli policy file:

Usage: ssscli policy file [OPTIONS] policy_name auth_obj_id%%CRLF
 SP%% Create Binary file object policy.%%CRLF
 SP%% policy_name = File name of the policy to be created. This policy name should be given as input while provisioning.%%CRLF
 SP%% auth_obj_id = Auth object id for each Object Policy.

Options:

--write BOOLEAN	Object policy Allow Write. Enabled by Default
--read BOOLEAN	Object policy Allow Read. Enabled by Default
--forbid_all BOOLEAN	Object policy forbid all. Disabled by Default
--delete BOOLEAN	Object policy Allow Delete. Enabled by Default
--req_sm BOOLEAN	Object policy Require Secure Messaging. Disabled by Default
--req_pcr_val BOOLEAN	Object policy Require PCR Value. Disabled by Default
--pcr_obj_id TEXT	Object policy PCR object ID in HEX format. Zero by Default
--pcr_expected_value TEXT	Object policy PCR Expected in Hex byte string Value. Zero by Default
--help	Show this message and exit.

4. ssscli policy counter:

Usage: ssscli policy counter [OPTIONS] policy_name auth_obj_id%%CRLF
 SP%% Create Counter object policy.%%CRLF
 SP%% policy_name = File name of the policy to be created. This policy name should

```
be given as input while provisioning.%%CRLF
SP%% auth_obj_id = Auth object id for each Object Policy.
```

Options:

--write BOOLEAN	Object policy Allow Write. Enabled by Default
--read BOOLEAN	Object policy Allow Read. Enabled by Default
--forbid_all BOOLEAN	Object policy forbid all. Disabled by Default
--delete BOOLEAN	Object policy Allow Delete. Enabled by Default
--req_sm BOOLEAN	Object policy Require Secure Messaging. Disabled by Default
--req_pcr_val BOOLEAN	Object policy Require PCR Value. Disabled by Default
--pcr_obj_id TEXT	Object policy PCR object ID in HEX format. Zero by Default
--pcr_expected_value TEXT	Object policy PCR Expected in Hex byte string Value. Zero by Default
--help	Show this message and exit.

5. ssscli policy userid:

```
Usage: ssscli policy userid [OPTIONS] policy_name auth_obj_id%%CRLF
SP%% Create user id object policy.%%CRLF
SP%% policy_name = File name of the policy to be created. This policy name
should
be given as input while provisioning.%%CRLF
SP%% auth_obj_id = Auth object id for each Object Policy.
```

Options:

--write BOOLEAN	Object policy Allow Write. Enabled by Default
--forbid_all BOOLEAN	Object policy forbid all. Disabled by Default
--delete BOOLEAN	Object policy Allow Delete. Enabled by Default
--req_sm BOOLEAN	Object policy Require Secure Messaging. Disabled by Default
--req_pcr_val BOOLEAN	Object policy Require PCR Value. Disabled by Default
--pcr_obj_id TEXT	Object policy PCR object ID in HEX format. Zero by Default
--pcr_expected_value TEXT	Object policy PCR Expected in Hex byte string Value. Zero by Default
--help	Show this message and exit.

6. ssscli policy pcr:

```
Usage: ssscli policy pcr [OPTIONS] policy_name auth_obj_id%%CRLF
SP%% Create PCR object policy.%%CRLF
SP%% policy_name = File name of the policy to be created. This policy name
should
be given as input while provisioning.%%CRLF
SP%% auth_obj_id = Auth object id for each Object Policy.
```

Options:

--write BOOLEAN	Object policy Allow Write. Enabled by Default
--read BOOLEAN	Object policy Allow Read. Enabled by Default
--forbid_all BOOLEAN	Object policy forbid all. Disabled by Default
--delete BOOLEAN	Object policy Allow Delete. Enabled by Default
--req_sm BOOLEAN	Object policy Require Secure Messaging. Disabled by Default
--req_pcr_val BOOLEAN	Object policy Require PCR Value. Disabled by Default
--pcr_obj_id TEXT	Object policy PCR object ID in HEX format. Zero by Default

```
--pcr_expected_value TEXT Object policy PCR Expected in Hex byte string
Value. Zero by Default
--help Show this message and exit.
```

7. ssscli policy dump:

```
Usage: ssscli policy dump [OPTIONS] policy_name%%CRLF
SP%% Display Created object policy.%%CRLF
SP%% policy_name = File name of the policy to be displayed.
```

Options:

```
--help Show this message and exit.
```

9.8 CLI Data formats

9.8.1 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:

```
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 462e ... .O-p....UFF.
00000030: 2cca 12e0 a046 90a3 3bea 8252 eeb5 4ff5 ,....F..;...R..O.
00000040: a4e9 b2d6 a144 0342 0004 633e 938a 67a7 .....D.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) -----
```

9.8.2 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-----
MIGHAgEAMBMGByqGSM49AgEGCCqGSM49AwEHBG0wawIBAQgCE8tc07kCblVRkYu
LMoS4KBGkKM76oJS7rVP9aTpstahRANCAARjPpOKZ6ep5vNfw2k7UTyIag3CYBbz
XowsC0kbI5J68DcdKLEb++j0LpxbDyiXxRbsPn1il+TAb5Ntup2sirzY
-----END PRIVATE KEY-----
```

9.8.3 HEX

Hex or [Hexadecimal](#) (base 16) is a positional system that represents numbers using a base of 16.

Example:

```
308187020100301306072a8648ce3d020106082a8648ce3d030107046d306b02  
01010420084f2d70eee409b95546462e2cca12e0a04690a33bea8252eeb54ff5  
a4e9b2d6a14403420004633e938a67a7a9e6f35fc3693b513c886a0dc26016f3  
5e8c2c0b491b23927af0371d28b11bfbe8f42e9c5b0f2897c516ec3e7d6297e4  
c06f936dba9dac8abcd8
```

9.8.4 REFERENCE KEY

- Refer to [Section 8.1.2.2EC Reference key format](#) for EC reference key format.
- Refer to [Section 8.1.2.3RSA Reference key format](#) for RSA reference key format.

9.9 Object Policies Through ssscli

Applying policy to objects through ssscli shall be done in two steps.

- Create object policy
- Attach policy to object

9.9.1 Create object policy

Object policy shall be created using following command:

```
ssscli policy
```

The create command has symkey, asymkey, userid, file, counter and pcr sub commands.

- symkey -> Symmetric key object policy (AES, DES, HMAC)
- asymkey -> Asymmetric key object policy (RSA, EC)
- userid -> User ID Object Policy
- file -> Binary file Object Policy
- counter -> Counter Object Policy
- pcr -> PCR Object Policy

Each command has mandatory arguments for policy_name and auth_object_id.

- policy_name -> Name of the policy to be created. This policy name should be given as input while provisioning.
- auth_object_id -> Auth object id for each Object Policy

Create policy command shall have following optional arguments based on the sub command selected and applet version of secure element:

- --sign -> Object policy Allow Sign. Enabled by Default. Parameter type is boolean.
- --verify -> Object policy Allow Verify. Enabled by Default. Parameter type is boolean.
- --encrypt -> Object policy Allow Encryption. Enabled by Default. Parameter type is boolean.
- --decrypt -> Object policy Allow Decryption. Enabled by Default. Parameter type is boolean.
- --key_derive -> Object policy Allow Key Derivation. Disabled by Default. Parameter type is boolean.
- --wrap -> Object policy Allow Wrap. Disabled by Default. Parameter type is boolean.
- --generate -> Object policy Allow Generate. Enabled by Default. Parameter type is boolean.
- --write -> Object policy Allow Write. Enabled by Default. Parameter type is boolean.
- --read -> Object policy Allow Read. Enabled by Default. Parameter type is boolean.
- --import_export -> Object policy Allow Import Export. Disabled by Default. Parameter type is boolean.

- --key_agreement -> Object policy Allow Key Agreement. Disabled by Default. Parameter type is boolean.
- --attest -> Object policy Allow attestation. Disabled by Default. Parameter type is boolean.
- --desfire_auth -> Object policy Allow to perform DESFire authentication. Disabled by Default. Parameter type is boolean.
- --desfire_dump -> Object policy Allow to dump DESFire session keys. Disabled by Default. Parameter type is boolean.
- --forbid_derived_output -> Object policy forbid derived output. Disabled by Default. Parameter type is boolean.
- --kdf_ext_random -> Object policy key derivation external random. Disabled by Default. Parameter type is boolean.
- --tls_kdf -> Object policy Allow tls kdf. Disabled by Default. Parameter type is boolean.
- --tls_pms_kd -> Object policy Allow tls pms kd. Disabled by Default. Parameter type is boolean.
- --hkdf -> Object policy Allow hkdf. Enabled by Default. Parameter type is boolean.
- --pbkdf -> Object policy Allow pbkdf. Disabled by Default. Parameter type is boolean.
- --desfire_kd -> Object policy Allow desfire kd. Disabled by Default. Parameter type is boolean.
- --forbid_external_iv -> Object policy forbid external IV. Disabled by Default. Parameter type is boolean.
- --usage_hmac_pepper -> Object policy Allow usage hmac as pepper. Disabled by Default. Parameter type is boolean.
- --desfire_change_key -> Object policy Allow desfire change key. Disabled by Default. Parameter type is boolean.
- --derived_input -> Object policy Allow derived input. Disabled by Default. Parameter type is boolean.
- --desfire_auth_id -> 32 bit desfire auth id for desfire_change_key policy. Parameter type is hexdecimal.
- --source_key_id -> 32 bit source key id for derived_input policy. Parameter type is hexdecimal.
- --forbid_all -> Object policy forbid all. Disabled by Default. Parameter type is boolean.
- --delete -> Object policy Allow Delete. Enabled by Default. Parameter type is boolean.
- --req_sm -> Object policy Allow req_sm. Disabled by Default. Parameter type is boolean.
- --req_pcr_val -> Object policy Require PCR Value. Disabled by Default. Parameter type is boolean.
- --pcr_obj_id -> Object policy PCR object ID. Zero by Default. Parameter type is hexdecimal.
- --pcr_expected_value -> Object policy PCR Expected Value. Zero by Default. Parameter type is hexdecimal byte array.

The created object policy stored in the system in pickle file format.

Command Sample:

```
(venv) C:\_ddm\simw-top\pycli>ssscli policy asymkey --help
Usage: ssscli policy asymkey [OPTIONS] policy_name auth_obj_id

    Create Asymmetric key object policy.

    policy_name = File name of the policy to be created. This policy name should
    be given as input while provisioning.

    auth_obj_id = Auth object id for each Object Policy.

    Options:
        --sign BOOLEAN          Object policy Allow Sign. Enabled by Default
        --verify BOOLEAN         Object policy Allow Verify. Enabled by
                                Default
        --encrypt BOOLEAN        Object policy Allow Encryption. Enabled by
                                Default
        --decrypt BOOLEAN        Object policy Allow Decryption. Enabled by
                                Default
        --key_derive BOOLEAN     Object policy Allow Key Derivation. Disabled
                                by Default
        --wrap BOOLEAN           Object policy Allow Wrap. Disabled by
                                Default
        --generate BOOLEAN       Object policy Allow Generate. Enabled by
                                Default
        --write BOOLEAN          Object policy Allow Write. Enabled by
                                Default
        --read BOOLEAN           Object policy Allow Read. Enabled by Default
        --import_export BOOLEAN  Object policy Allow Import Export. Disabled
                                by Default
        --key_agreement BOOLEAN Object policy Allow Key Agreement. Disabled
                                by Default
        --attest BOOLEAN         Object policy Allow attestation. Disabled by
                                Default
        --forbid_derived_output BOOLEAN
                                Object policy Forbid Derived Output.
                                Disabled by Default
        --forbid_all BOOLEAN     Object policy forbid all. Disabled by
                                Default
        --delete BOOLEAN         Object policy Allow Delete. Enabled by
                                Default
        --req_sm BOOLEAN         Object policy Require Secure Messaging.
                                Disabled by Default
        --req_pcr_val BOOLEAN   Object policy Require PCR Value. Disabled by
                                Default
        --pcr_obj_id TEXT        Object policy PCR object ID in HEX format.
                                Zero by Default
        --pcr_expected_value TEXT Object policy PCR Expected in Hex byte
                                string Value. Zero by Default
        --help                  Show this message and exit.
```

Usage example:

```
ssscli policy asymkey ecc_sign_policy 0x7DA00001 --sign 0
```

Created object policy shall be displayed using following command:

```
ssscli policy dump <policy_name>
```

Usage example:

```
(venv) C:\_ddm\simmw-top\pycli>ssscli policy dump ecc_sign_policy
Reading policy from file path:
c:\_ddm\simmw-top\pycli\policy\ssscli_obj_policy_ecc_sign_policy.pkl

Created object policy:
Middleware compiled applet version: 6.12

Key_type : Asymmetric_Key
Auth Obj ID : 0x7DA00001
Sign : False
Verify : True
Encrypt : True
Decrypt : True
Key Derive : False
Wrap : False
Generate : True
Import Export : False
Key Agreement : False
Attestation : False
forbid_derived_output : False
Write : True
Read : True
forbid_all : False
Delete : True
req_sm : False
pcr_obj_id : 0x0
pcr_expected_value :

Policy in hex:

08 7D A0 00 01 0B 3C 00 00
```

9.9.2 Attach policy to object

Created object policy shall be applied it to the object along with generate or set command using --policy_name optional parameter.

Command Sample:

```
(venv) C:\workspace\iot\simmw-top\pycli>ssscli generate ecc --help
Usage: ssscli generate ecc [OPTIONS] keyid [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]

Generate ECC Key

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:
--policy_name TEXT  File name of the policy to be applied
--help              Show this message and exit.
```

Usage example:

```
ssscli generate ecc 0x20181001 NIST_P256 --policy_name ecc_sign_policy
ssscli set ecc pair 0x20182010 nistp521_key.pem --policy_name ecc_sign_policy
```

9.10 Upload keys and certificates to SE05X using ssscli tool

1. Refer to [CLI Tool](#) for ssscli tool setup
2. Keys and Certificates can be uploaded to SE05X using the ssscli tool

```
ssscli connect se05x 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

10 Appendix

10.1 Glossary

Table 4. Glossary

Term	Definition
EAR	Early Access Release
SE	Secure Element

10.2 VCOM

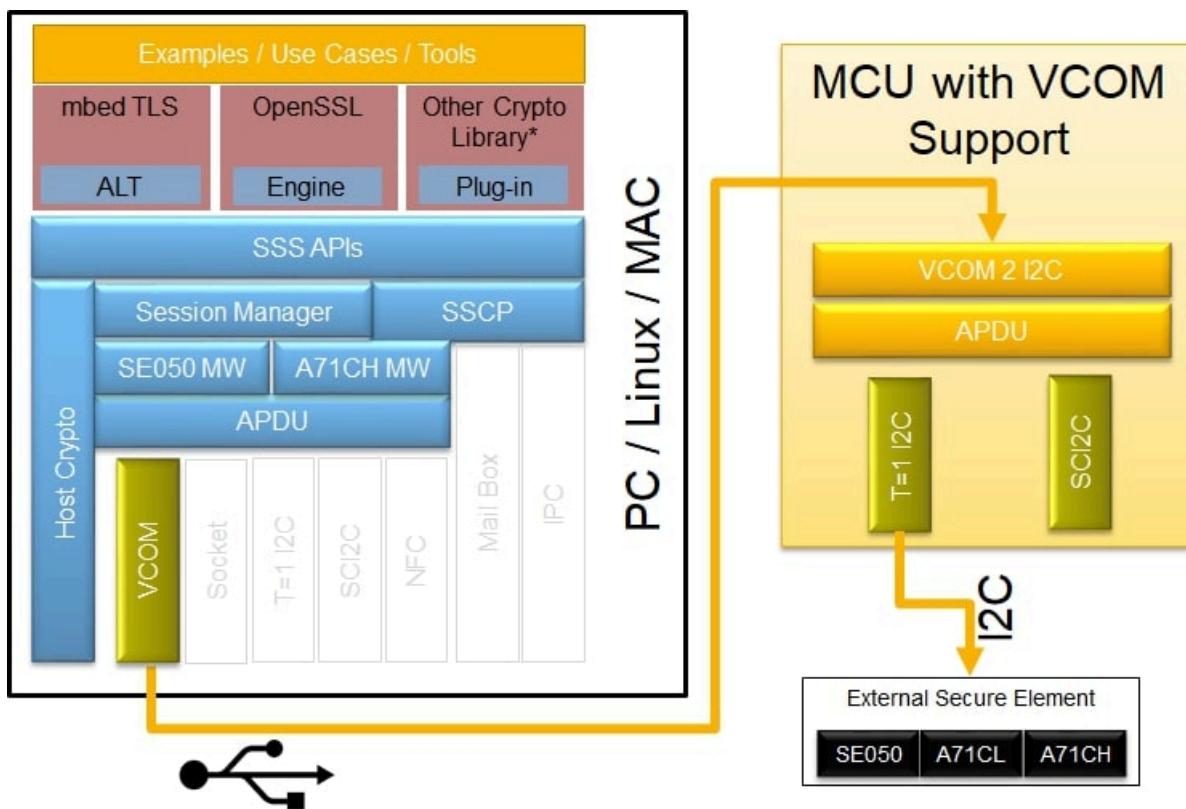
Virtual COM Port interface can be used to connect to the SE from development PC/Laptop. This gives advantage of faster turn around time for development and feature experimentation.

Note:

Drivers

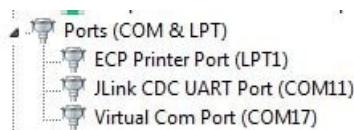
If the VCOM drivers are not installed on your PC, you would have to install it.

Also, for embedded platforms like [Freedom K64F](#), to run [CLI Tool](#)



10.2.1 Steps to use VCOM for running the examples - PC & FRDM-K64F

- Connect FRDM-K64F with, SE050 attached to it on, the USB Interface (Not the OpenSDA USB).
- FRDM-K64F should have the VCOM firmware. If not,
- Import se05x_vcom project into MCUXpresso. Build and download the VCOM firmware. For details on MCUXpresso Build and Debug,
- Open device manager, Under Ports (COM & LPT), look for the connected device.



- specify this COMport for connecting from PC. For e.g. COM1 7 as per the above image.

Note:

It appears as “Virtual Com Port” or “USB Serial Device” based on the driver installed in the system.

10.3 APDU Commands over VCOM

Sending and Receiving Applet APDU Commands on VCOM using a Serial Terminal Emulator.

10.3.1 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

10.3.2 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"> 00: SoftReset / Fetch ATR 01: Other commands
2	Node Address
3	Len MSB
4	Len LSB

10.3.3 Example Commands

The example shows the commands strings to be sent over VCOM and some parsing of the commands/ responses

10.3.3.1 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 00 00 00 00 64 00 00 0A 4A 43 4F 50 34 20 41 54 50 4F
VCOM Header	00 00 00 23

Field	Bytes in Hex
Atr Response	-- -- -- -- 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

10.3.3.2 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 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 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 Response	-- -- -- -- 03 01 00 3F FF 01 0B 90 00
applet Version	appletVersion in TLV with Tag1, (Len=7) -- -- -- -- -- -- 03 01 00 3F FF 01 0B

10.3.3.3 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 Response	-- -- -- -- 41 82 00 07 03 01 00 3F FF 01 0B 90 00
applet Version	appletVersion in TLV with Tag1, (Len=7) -- -- -- -- -- -- 03 01 00 3F FF 01 0B

10.3.3.4 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 Response	-- -- -- -- 41 82 00 08 B5 47 3C 8C A5 16 AC 31 90 00

Field	Bytes in Hex
RandomData	-- -- -- -- -- -- -- -- B5 47 3C 8C A5 16 AC 31

10.3.3.5 GetUID

UID is an object with Object ID = 0x7FFF0206.

Field	Bytes in Hex
Command	01 00 00 13 80 02 00 00 00 00 0A 41 04 7F FF 02 06 43 02 00 12 00 00
VCOM Header	01 00 00 13
APDU	-- -- -- -- 80 02 00 00 00 00 0A 41 04 7F FF 02 06 43 02 00 12 00 00
Response	01 00 00 18 41 82 00 12 04 00 50 01 55 55 55 55 55 55 55 04 FF FF FF FF FF FA 90 00
VCOM Header	01 00 00 18
APDU Response	-- -- -- -- 41 82 00 12 04 00 50 01 55 55 55 55 55 55 55 55 04 FF FF FF FF FF FA 90 00
UID	UID in TLV with Tag1 -- -- -- -- -- -- -- 04 00 50 01 55 55 55 55 55 55 55 55 04 FF FF FF FF FF FA

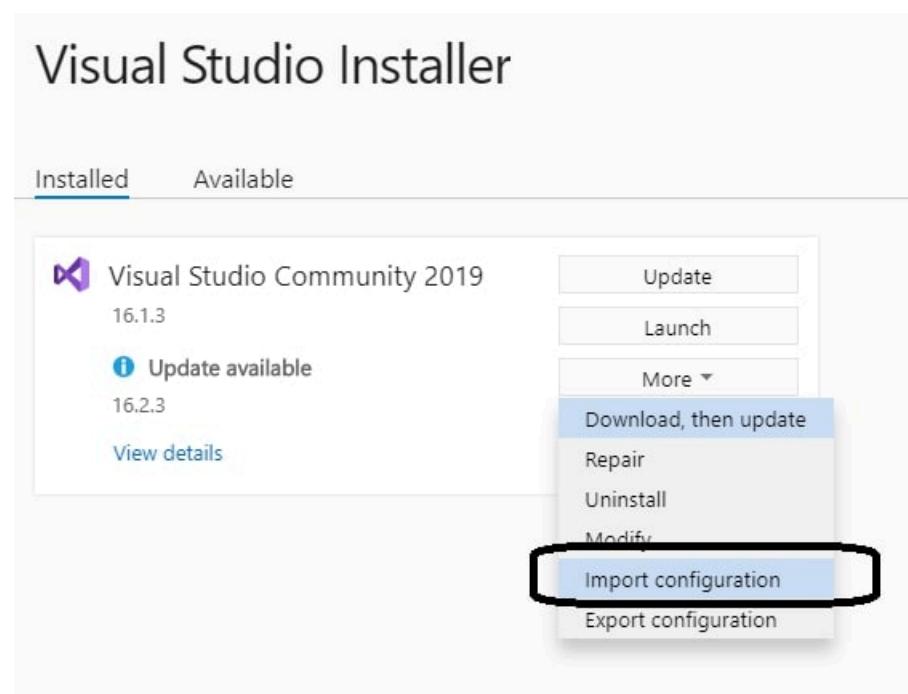
10.4 Visual Studio 2019 Setup

10.4.1 Prerequisites

Install any edition of Visual Studio 2019, It will install the visual studio installer application.

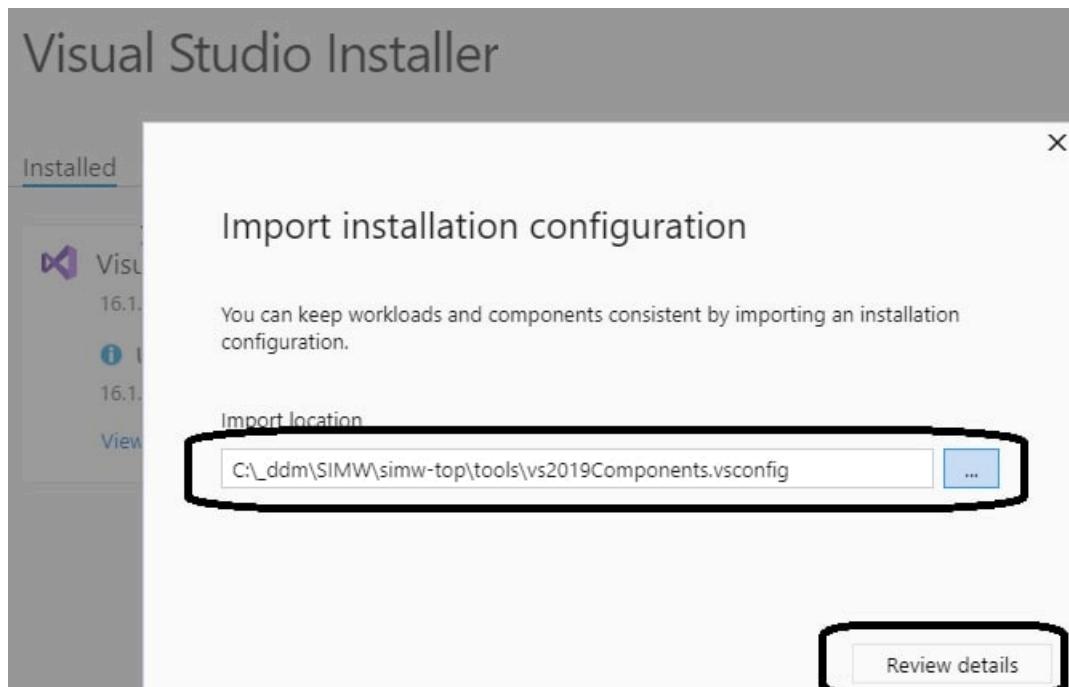
10.4.2 Installing the components

1. Open Visual Studio Installer, click on more and then import configuration.



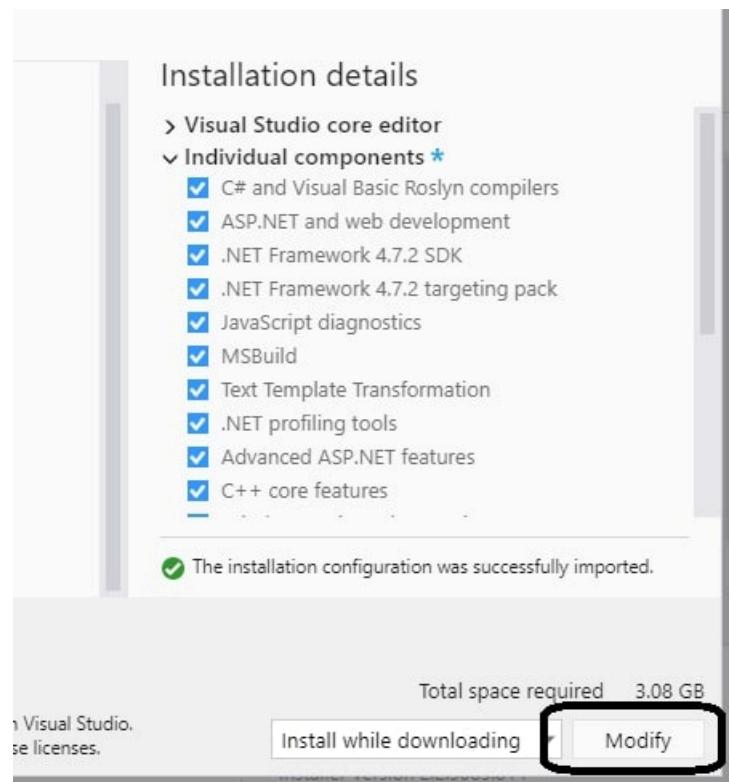
2.

- In the dialogue box give the path to the configuration file present in tools folder.
tools/vs2019Components.vsconfig



3.

- click on modify.



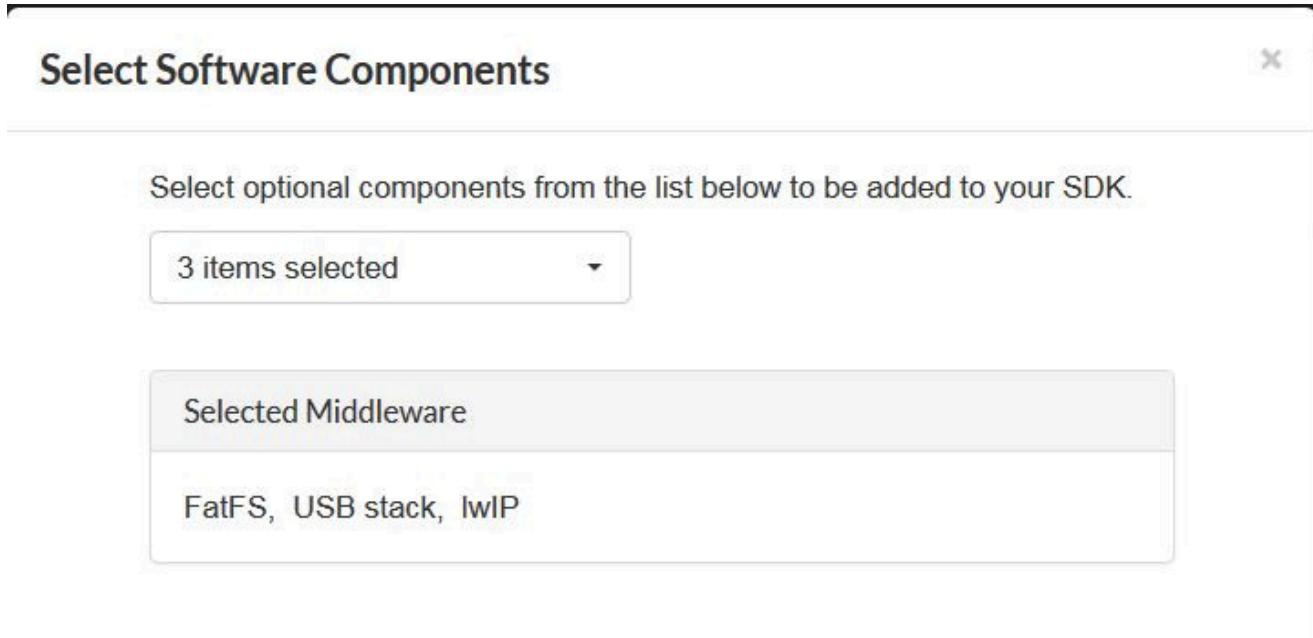
10.5 Setting up MCUXpresso IDE

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

10.5.1 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 EVK-MIMXRT1060.
- Click on Build MCUXpresso SDK.
- Select Software Components.

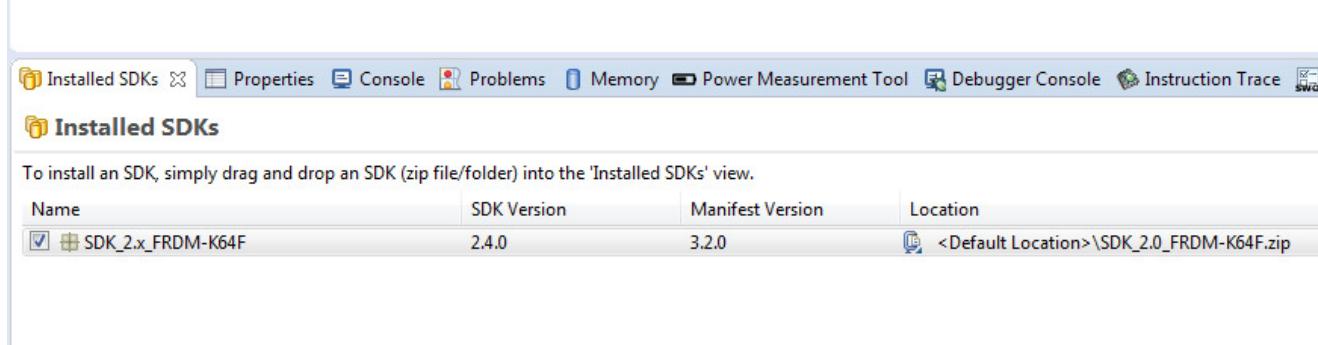
**Note:**

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

10.5.2 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.



10.6 Development Platforms

10.6.1 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

10.6.1.1 Downloading and Installing Yocto for MCIMX8M-EVK

Please consult first the detailed information that can be found on <https://www.nxp.com>. Download the **i.MX Yocto Project User's Guide** document as a starting point(available on https://www.nxp.com/design/software/embedded-software/i-mx-software/embedded-linux-for-i-mx-applications-processors:IMXLINUX?tab=Documentation_Tab under the **Release and Documentation** header).

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-6.6.52_2.2.0
cd ~/projects/imx-yocto-bsp-6.6.52_2.2.0
repo init -u https://github.com/nxp-imx/imx-manifest -b imx-linux-scarthgap -m
imx-6.6.52-2.2.0.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-6.6.52_2.2.0/sources
cd ~/projects/imx-yocto-bsp-6.6.52_2.2.0/sources
tar xzvf meta-custom.tgz
```

10.6.1.2 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-6.6.52_2.2.0
DISTRO=fsl-imx-wayland MACHINE=imx8mqevk source imx-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 = "7"
BBPATH = "${TOPDIR}"
```

```

BSPDIR := "${@os.path.abspath(os.path.dirname(dgetVar('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-openembedded/meta-python \
\
${BSPDIR}/sources/meta-freescale \
${BSPDIR}/sources/meta-freescale-3rdparty \
${BSPDIR}/sources/meta-freescale-distro \
"
# i.MX Yocto Project Release layers
BBLAYERS += "${BSPDIR}/sources/meta-imx/meta-imx-bsp"
BBLAYERS += "${BSPDIR}/sources/meta-imx/meta-imx-sdk"
BBLAYERS += "${BSPDIR}/sources/meta-imx/meta-imx-ml"
BBLAYERS += "${BSPDIR}/sources/meta-imx/meta-imx-v2x"
BBLAYERS += "${BSPDIR}/sources/meta-nxp-demo-experience"
BBLAYERS += "${BSPDIR}/sources/meta-nxp-connectivity/meta-nxp-matter-baseline"
BBLAYERS += "${BSPDIR}/sources/meta-nxp-connectivity/meta-nxp-openthread"

BBLAYERS += "${BSPDIR}/sources/meta-arm/meta-arm"
BBLAYERS += "${BSPDIR}/sources/meta-arm/meta-arm-toolchain"
BBLAYERS += "${BSPDIR}/sources/meta-clang"
BBLAYERS += "${BSPDIR}/sources/meta-openembedded/meta-gnome"
BBLAYERS += "${BSPDIR}/sources/meta-openembedded/meta-networking"
BBLAYERS += "${BSPDIR}/sources/meta-openembedded/meta-filesystems"
BBLAYERS += "${BSPDIR}/sources/meta-qt6"
BBLAYERS += "${BSPDIR}/sources/meta-security/meta-parsec"
BBLAYERS += "${BSPDIR}/sources/meta-security/meta-tpm"
BBLAYERS += "${BSPDIR}/sources/meta-virtualization"

# +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 dev-pkgs tools-debug tools-sdk tools-testapps package-management"
USER_CLASSES ?= "buildstats"
PATCHRESOLVE = "noop"
BB_DISKMON_DIRS ??= "\
    STOPTASKS, ${TMPDIR}, 1G, 100K \
    STOPTASKS, ${DL_DIR}, 1G, 100K \
    STOPTASKS, ${SSTATE_DIR}, 1G, 100K \
    STOPTASKS, /tmp, 100M, 100K \
    HALT, ${TMPDIR}, 100M, 1K \
    HALT, ${DL_DIR}, 100M, 1K \
    HALT, ${SSTATE_DIR}, 100M, 1K \
    HALT, /tmp, 10M, 1K"

```

```

PACKAGECONFIG:append:pn-qemu-system-native = " sdl"
PACKAGECONFIG:append:pn-nativesdk-qemu = " sdl"
CONF_VERSION = "2"

DL_DIR ?= "${BSPDIR}/downloads/"
ACCEPT_FSL_EULA = "1"

# +SIMW
UBOOT_CONFIG = "sd"
UBOOT_CONFIG[sd] = "imx8mq_evk_config_sdcard"
IMAGE_FSTYPES:append = " wic.bz2"
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 "
# opca
IMAGE_INSTALL:append = " python3-logging python3-core"

IMAGE_INSTALL:append = " e2fsprogs-resize2fs func-timeout "
IMAGE_INSTALL:append = " i2c-tools "
IMAGE_INSTALL:append = " python3-misc"
IMAGE_INSTALL:append = " opensc "
# -SIMW
IMAGE_INSTALL:append = " se05x"

```

Note:

At this stage it's possible to also include the Plug&Trust package as a Yocto recipe. Refer to [Alternative approach to create SD card: use Yocto recipe for Plug&Trust package](#) on how to do this.

The above local.conf file prepares an embedded Linux distribution with native development tools

Note:

If i.MX 8M Quad EVK Board (old version) is used then change UBOOT_CONFIG[sd] = "imx8mq_wevk_config_sdcard" to UBOOT_CONFIG[sd] = "imx8mq_evk_config_sdcard" Comment the line IMX_DEFAULT_BSP = "nxp" in imx-yocto-bsp-6.6.52_2.2.0/sources/meta-freescale/conf/machine/imx8mq-lpddr4-wevk.conf

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-6.6.52_2.2.0/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.

10.6.1.3 Create and install SDK and Root file system on Host

To populate SDK, run:

```
cd ~/projects/imx-yocto-bsp-6.6.52_2.2.0
MACHINE=imx8mqevk source setup-environment build-wayland-imx8mqevk
bitbake -c populate_sdk core-image-full-cmdline
```

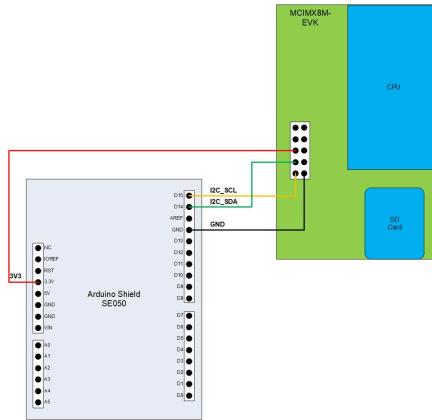
To install the SDK in the default location `/opt/fsl-imx-wayland/4.19-warrior`:

```
cd ~/projects/imx-yocto-bsp-6.6.52_2.2.0/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.

10.6.1.4 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

10.6.1.5 Alternative approach to create SD card: use Yocto recipe for Plug&Trust package

The Plug&Trust MW package also includes a recipe.

- Download the Plug&Trust MW from www.nxp.com/se050 and rename the package to SE050-PLUG-TRUST-MW.zip
- Put the se05x recipe (se05x_mw.bb), the source code package (SE050-PLUG-TRUST-MW.zip) and (if applicable) patches into the existing .../sources/meta-custom/recipes directory of the Yocto development PC (recipe and patches are available in simw-top/scripts/yocto/bb_conf_files)

The resulting imx-yocto-bsp-6.6.52_2.2.0/sources/meta-custom directory on the Yocto development PC should look like:

```
└── conf
    └── layer.conf
── README
└── recipes
    ├── func-timeout
    │   └── func-timeout_4.3.5.bb
    └── se05x
        ├── files
        │   └── SE050-PLUG-TRUST-MW.zip
        └── se05x_mw.bb
```

To add the Plug&Trust MW to the image to be created add the following line at the end of the build-wayland-imx8mqevk/conf/local.conf file:

```
IMAGE_INSTALL_append += " se05x"
```

10.6.2 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.

10.6.2.1 Downloading and Installing Yocto for MCIMX6UL-EVK

Please consult first the detailed information that can be found on <https://www.nxp.com>. Download the **i.MX Yocto Project User's Guide** document as a starting point(available on https://www.nxp.com/design/software/embedded-software/i-mx-software/embedded-linux-for-i-mx-applications-processors:IMXLINUX?tab=Documentation_Tab under the **Release and Documentation** header).

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-6.6.52_2.2.0
cd ~/projects/imx-yocto-bsp-6.6.52_2.2.0
repo init -u https://github.com/nxp-imx/imx-manifest -b imx-linux-scarthgap -m imx-6.6.52-2.2.0.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-6.6.52_2.2.0/sources
cd ~/projects/imx-yocto-bsp-6.6.52_2.2.0/sources
tar xzvf meta-custom.tgz
```

10.6.2.2 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-5-4-70-2-3-4
DISTRO=fsl-imx-fb MACHINE=imx6ulevk source imx-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(dgetVar('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 += "${BSPDIR}/sources/meta-qt5"
# +SIMW
BBLAYERS += "${BSPDIR}/sources/meta-custom"
# -SIMW
```

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"
# + 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"

DL_DIR ??= "${BSPDIR}/downloads/"
ACCEPT_FSL_EULA = "1"%%CRLF
CRLF%# +SIMW
IMAGE_ROOTFS_EXTRA_SPACE = "640000"

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"
# -SIMW
```

Note:

At this stage it's possible to also include the Plug&Trust package as a Yocto recipe. Refer to [Alternative approach to create SD card: use Yocto recipe for Plug&Trust package](#) on how to do this.

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 a compressed sdcard image containing bootloader, filesystem and linux kernel is available under `~/projects/imx-yocto-bsp-5-4-70-2-3-4/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.

10.6.2.3 Create and install SDK and Root file system on Host

To populate SDK, run:

```
cd ~/projects/imx-yocto-bsp-5-4-70-2-3-4  
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/5.4.70-2.3.4`:

```
cd ~/projects/imx-yocto-bsp-5-4-70-2-3-4/build-fb-6ul/tmp/deploy/sdk/  
.fsl-imx-fb-glibc-x86_64-core-image-full-cmdline-cortexa7hf-neon-toolchain-5.4.  
70-2.3.4.sh
```

You have now installed a set of cross-compilation tools for the MCIMX6UL-EVK board on the Linux Host PC.

10.6.2.4 Alternative approach to create SD card: use Yocto recipe for Plug&Trust package

The Plug&Trust MW package also includes a recipe.

- Download the Plug&Trust MW from www.nxp.com/se050 and ensure the package version (e.g. 04.00.00) is appended to the file name. (For example: SE050-PLUG-TRUST-MW-v04.00.00.zip)
- Put the se05x recipe (`se05x_4.0.0.bb`), the source code package (SE050-PLUG-TRUST-MW-v04.00.00.zip) and (if applicable) patches into the existing `.../sources/meta-custom/recipes` directory of the Yocto development PC (recipe and patches are available in `simw-top/scripts/yocto/v04.00.00`)

The resulting `imx-yocto-bsp-5-4-70-2-3-4/sources/meta-custom` directory on the Yocto development PC should look like:

```
+- conf  
|   +- layer.conf  
+- README  
+- recipes  
    +- func-timeout  
    |   +- func-timeout_4.3.3.bb  
    +- se05x  
        +- files  
        |   +- SE050-PLUG-TRUST-MW-v04.00.00.zip  
    +- se05x_4.0.0.bb
```

To add the Plug&Trust MW to the image to be created add the following line at the end of the build-wayland-imx8mqevk/conf/local.conf file:

```
IMAGE_INSTALL_append += " se05x"
```

10.6.3 Freedom K64F

See [Section 4.2 — Import MCUXPresso projects from SDK](#) and [Section 4.3 — Freedom K64F \(and Other MCUs\) Build \(CMake - Advanced\)](#)

10.6.4 i.MX RT 1060

See [Section 4.3.2 Importing the Project](#).

10.6.4.1 MIMXRT1060 Boot Mode and Boot Device Settings

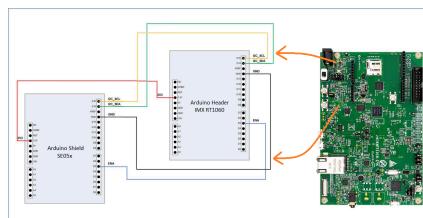
SW7-1	SW7-2	SW7-3	SW7-4	Boot Device
OFF	ON	ON	OFF	Hyper flash
OFF	OFF	ON	OFF	QSPI flash
ON	OFF	ON	OFF	SD card

Note:

Default/Current Settings

Boot Device - QSPI flash Flash driver - MIMXRT1060_SFDP_QSPI.cfx

10.6.4.2 Connecting SE050 Arduino shield to i.MX RT 1060



10.6.5 LPC55S69

See [Section 4.3.2 Importing the Project](#).

10.6.5.1 Connecting SE050 Arduino shield to LPC55S

Connect SE050 to LPC55S 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 Silex-2401 SPI pins .. so we cannot use SE_EN signal to drive SE_VDD.

10.6.5.2 Connecting WiFi shield to LPC55S

Connect the muRata CMWC1ZZABR-107-EVB WiFi shield to LPC55S mikroBUS as:



Note:

Maximum of 4 connections are supported at a time by the shield. Further connection attempts will result in failure.

10.6.5.3 Memory regions

Secure Zone

Total code size available for secure applications is 0x3FE00 bytes.

Memory area	Origin	Size
m_interrupts	0x10000000	LENGTH = 0x00000200
m_text	0x10000200	LENGTH = 0x0003FC00
m_veneer_table	0x1003FE00	LENGTH = 0x00000200

Memory area	Origin	Size
m_core1_image	0x1007A000	LENGTH = 0x0001E000
m_data	0x30000000	LENGTH = 0x00018000
rpmsg_sh_mem	0x30033000	LENGTH = 0
m_usb_sram	0x50100000	LENGTH = 0x00004000

Non-secure Zone

Total code size available for non-secure applications is 0x3A000 bytes.

Memory area	Origin	Size
m_interrupts	0x00040000	LENGTH = 0x00000200
m_text	0x00040200	LENGTH = 0x00039E00
m_core1_image	0x0007A000	LENGTH = 0x0001E000
m_data	0x20018000	LENGTH = 0x0001A800
rpmsg_sh_mem	0x20033000	LENGTH = 0
m_usb_sram	0x40100000	LENGTH = 0x00004000

Note:

While modifying Secure/Non-secure memory regions, be sure that code size is 32kB aligned.

10.6.6 MIMXRT1170

See [Section 4.3.2 Importing the Project](#).

10.6.6.1 MIMXRT1170 Boot Mode and Boot Device Settings

Configure on-board switches for QSPI boot mode:

Boot source	SW1				SW2											
QSPI Flash	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

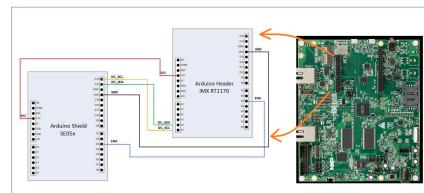
Note:

Default/Current Settings

Boot Device - QSPI flash

Flash driver - MIMXRT1170_SFDP_QSPI.cfx

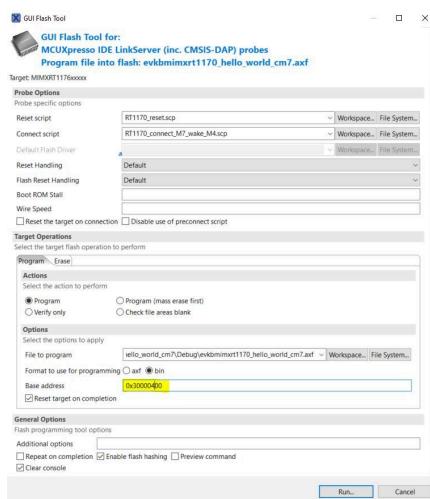
10.6.6.2 Connecting SE050 Arduino shield to i.MX RT 1170



10.6.6.3 Running / Debugging example

For more information on how to run the example, check [Import MCUXPresso projects from SDK](#).

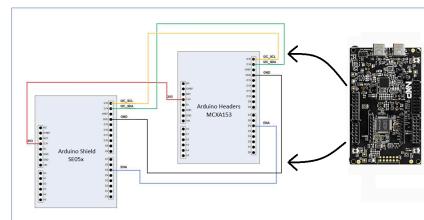
While flashing .bin file, update the base address to 0x30000400.



10.6.7 FRDM-MCXA153

See [Section 4.3.2 Importing the Project](#).

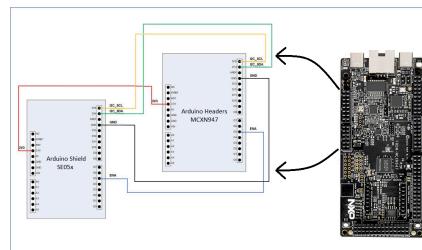
10.6.7.1 Connecting SE050 Arduino shield to FRDM-MCXA153



10.6.8 FRDM-MCXN947

See [Section 4.3.2 Importing the Project](#).

10.6.8.1 Connecting SE050 Arduino shield to FRDM-MCXN947



10.6.9 Android: Hikey960

10.6.9.1 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

10.6.9.2 AOSP build Environment Setup

10.6.9.2.1 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-  
review-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 Keystream 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 Keystream 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 bulding hikey kernel and follow the instruction given on link to create new bootimage image.

10.6.9.2.2 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). 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).

```
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.

- 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.
- 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.
```

10.6.9.3 SE050 based Android Keystarter

10.6.9.3.1 CMAKE based build system

- 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
```

- Once you are able to bring-up Android build environment for hikey960 follow below steps to build SE050 based android keystarter:

```
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/keystarter

Note:

If the patches are already applied, then instead of calling setup_script.sh, call build_script.sh

- A batch script keystarter_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.

10.6.9.3.2 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.

10.6.9.4 Extract Secure Element Information

Refer to [SE Platform Information on Android platform](#).

10.6.9.5 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.

10.6.9.6 How to use own Platform SCP03 Keys

Refer to [Section 10.11Using 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.

10.6.9.7 Retrieve Existing Certificates

Refer to [Get Certificate from the SE](#).

10.6.9.8 Key Attestation

Key attestation support is available for RSA and ECC keys. The attestation keys and certificates need to be pre-injected at the following keyIDs:

```
#define ATTESTATION_KEY_RSA_KEYID 0x00000001  
#define ATTESTATION_KEY_ECC_KEYID 0x00000004  
#define ATTESTATION_CERTIFICATE_RSA_KEYID 0x00000011  
#define ATTESTATION_CERTIFICATE_ECC_KEYID 0x00000014
```

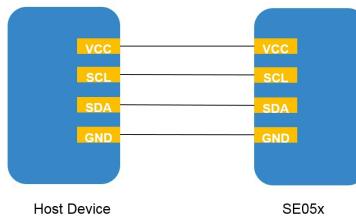
For details on how to inject certificates into SE, refer [Section 5.7.19Inject Certificate into SE](#).

10.6.9.9 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`

10.6.9.9.1 I2C connections with SE05x

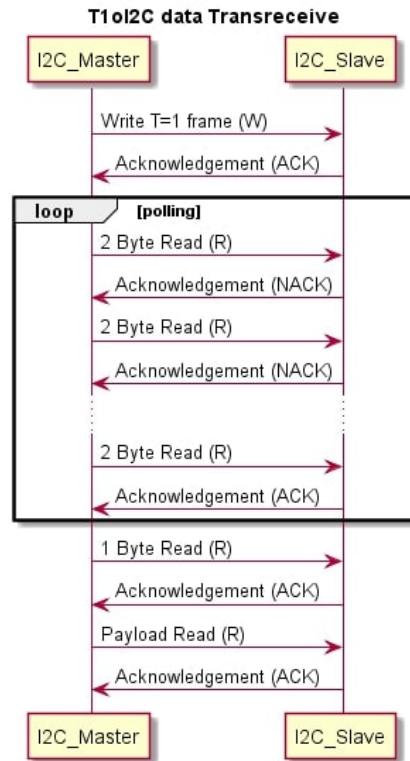
1. Below Diagram shows the wiring connection between Host Device and SE05x



10.6.9.9.2 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.

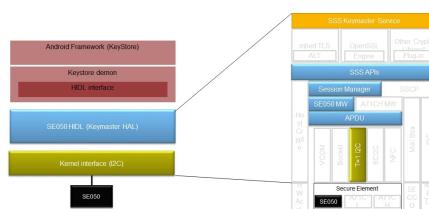
3. SE processes the received frame and prepares the response accordingly. HD polls for Read till the time Response is prepared and sent over I2C bus.
4. Following Diagram demonstrate the same.



10.6.9.10 Stack with SE050

We use Android Keystmaster 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:



10.6.9.11 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 keystmaster `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.

10.6.9.11.1 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:....  
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:....  
prime1:  
    xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:....  
prime2:  
    xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:....  
exponent1:  
    xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:....  
exponent2:  
    xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:....  
coefficient:  
    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;
}

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-----
\nMIIB9zCCAWCgAwIBAgIEITKMnTANBgkqhkiG9w0BAQsFADA3MRgwFgYDVQQDEw9JbnR1cm5hbFVzZU9ubHkxGzAZ
+95LNOSG03eAyk5bRC0KvbJVZIw0Ru2jpQyGX1D1ASnPjD4InQLPg9qIvSz3U7m84PR5WLexq
+wyUiraMeWpXB1Ln8FVQoKNidMF+DhK7TaGWK2oMHFK/s14cnwE1/5Te5uunM71i2NPphTjbUMrUH7Du
+t/EWb0UdHN4mGKKQIDAQABoxAwDjAMBgNVHRMEBTADAQH/
MA0GCSqGSIb3DQEBCwUA4GBAAgMSkfLAVijSRRTAdPhqyk0ifNYDJRAqoJNrxibN2pvsaUFyo8kde7MhCnNNsHZSt
\n-----END CERTIFICATE-----";

    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;
}
```

10.6.9.11.2 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;
    ECParameterSpec ECPspec;
    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(ECParameterSpec.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;
}

return privKey;
}

void SetTPKeyEC(String Label) throws NoSuchAlgorithmException,
InvalidKeySpecException, KeyStoreException,
CertificateException, GeneralSecurityException {
ECPrivateKey privKey;
X509Certificate cCert;
final String cDummyCert = "-----BEGIN CERTIFICATE-----\n" +
"MIIBeTCCASCgAwIBAgIJAKtU6mCCLJeyMAoGCCqGSM49BAMCBExDzANBgNVBAMMBmR1bW9DQTAeFw0xOTA1MDMxN
++xuGvo9FnFp7Tp0vA+xrZ0oIy2bsuzFr1VCZt7J00B2E+kxMFPCISi/
wC3xvYYQv6o7116kcQxw9CByjYTBfMAwGA1UdEwQFMAMBAf8wTwYDVR0RBEGwRoEmZDpOWFAtQTcxQ0gtRDozNzc4M
PAkaVE/u/o5a1v4hJ7jLH06NvdaatTINz0umK/\n-----END CERTIFICATE----- ";

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;  
}
```

10.7 How to get SE Platform Information and UID

Follow any one of the following methods to get the SE platform information.

10.7.1 Using TeraTerm and pre-built binary

A pre-built binary for se05x_GetInfo is available in binaries directory for FRDM-K64F and iMX-RT1060.

You would need to install a serial terminal application, like TeraTerm.

- Install TeraTerm on your system. To setup TeraTerm, refer [here](#).
 - Flash the pre-built binary on your hardware.
 - View the logs on TeraTerm.

You would be able to see a log like this:

The highlighted log is the SE UID, OEF ID and JCOP Platform ID.

10.7.2 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_GetInfo` with the following configuration:**

- SMCOM: VCOM
 - Host: PCWindows

- Run the built binary as:

se05x_GetInfo.exe COMxx

Where COMxx is the VCOM port number obtained from step 2.

You would be able to see a log like this:

The highlighted log is the SE UID, OEF ID and JCOP Platform ID.

10.7.3 Using ssscli 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/PCWindows/sscli` 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:

The highlighted log is the SE UID

10.7.4 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:

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.

The highlighted log is the SE UID, OEF ID and JCOP Platform ID.

10.8 Version Information

Item	Version Number
Release Version	v04.07.01_20250519
Middleware	v04.07.01_20250519
SSS APIs	v04.07.01_20250519
Demos and Use Cases	v02.28.00_20250519
JCOP	Platform ID J3R351021E950400
SE050 Applet	03.01
A71CH Applet	1.3

10.9 Certificate Chains

10.9.1 SE050 Certificate Chains

10.9.1.1 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. More information on these certificates can be found in the application note on SE050 configurations: [AN12436](#)

10.9.1.1.1 ECC

This directory contains the ECC chain of trust for cloud on-boarding.

10.9.1.1.1.1 ROOT CA

The file `IOT_NXP-01-CERT_IOT_CA_KEY-IoTRootCAvE305-01-20190320162439-EC_SEC_P384R1-4B7E5A.crt` contains ROOT CA.

```
-----BEGIN CERTIFICATE-----  
MIIB1jCCAVmgAwIBAgIBATAMBggqhkJOPQQDAwUAMEExFzAVBgNVBAsMD1BsdWcg  
YW5kIFRydXN0MQwwCgYDVQQKDANOWFAxGDAWBgNVBAMMD05YUCBSb290Q0F2RTMw  
NTAeFw0xOTAzMjAxNTI2MDRaFw0zNzAzMjAxNTI2MDRaMEExFzAVBgNVBAsMD1Bs  
dWcgYW5kIFRydXN0MQwwCgYDVQQKDANOWFAxGDAWBgNVBAMMD05YUCBSb290Q0F2  
RTMwNTB2MBAGByqGSM49AgEGBSuBBAAiA2IABNRrkWrw7iwM3oTw1Ay8I1yzOF+g  
OTNFzqfn/93N1xcK8kNEA8wNuuvjzsDXKHx701jrGZfy7YLjKYTwZR5wURXrE7lp  
PIwwdWE3OokTmhiMiFXnzSgSHCgtb+VF6nv76aMjMCEwDwYDVR0TAQH/BAUwAwEB  
/zAOBgNVHQ8BAf8EBAMCAQYwDAYIKoZIzj0EAwMFAAnpADBmAjEA7cCFz6PcBHki  
HRtbE3qi0Lj9iqxe8/2w8XrLc1AYhpMzzOQC05j3ZmgJ22B1bLk3AjEAhVnhuawO  
Zh1Jqwf7zySh/rNJeZfhGTyHAjQ9tTfy9eZAdc25feEN4j/Ad+7TF1Rg  
-----END CERTIFICATE-----
```

10.9.1.1.1.2 Intermediate CA

The file `IOT_NXP-01-CERT_IOT_4LAYER_CA_KEY-IoTInt4LAYERCAvE205-01-20190320164314-EC_SEC_P256R1-14DBBE.crt` contains the Intermediate CA.

```
-----BEGIN CERTIFICATE-----  
MIIBYjCCAU6gAwIBAgIBBDAMBggqhkJOPQQDAgUAMEExFzAVBgNVBAsMD1BsdWcg  
YW5kIFRydXN0MQwwCgYDVQQKDANOWFAxGDAWBgNVBAMMD05YUCBSb290Q0F2RTMw  
NTAeFw0xOTAzMjAxNTQyNTRaFw0zNDazMjAxNTQyNTRaMFAxFzAVBgNVBAsMD1Bs  
dWcgYW5kIFRydXN0MQwwCgYDVQQKDANOWFAxJzALBgnVBAMMhk5YUCBJbnR1cm1l  
ZG1hdGUtNExheWVyQ0F2RTIwNTBZMBMGBYqGSM49AgEGCCqGSM49AwEHA0IABo+d  
HK3Oa4FnkKN9/1JQpR2KarpwmwNRaEG6Zb+kzTwnXRw4Cvnd8IAcjXHqb93Vfx5  
rO+4MjX/gzqYagJByWejJjAkMBIGA1UdEwEB/wQIMAYBAf8CAQEwDgYDVR0PAQH/  
BAQDAgEGMAwGCCqGSM49BAMCBQADaAAwZQIxAPvXuv1W+zksBbz0NyyzpgFP1rCj  
IZOHpfZhaERw/DEj+4aAESa5vgtiR3CspIP5pAIwRsD1Z9fdBnPslavMmNAXjAlZ  
FPhbV7A0OYdSYhI8M5uTENrdezvsYYr2jfqIEcp  
-----END CERTIFICATE-----
```

-----END CERTIFICATE-----

10.9.1.1.2 RSA

This directory contains the RSA chain of trust for cloud on-boarding.

10.9.1.1.2.1 ROOT CA

The file IOT_NXP-01-CERT_IOT_CA_KEY-IoTRootCAvR406-01-20190425163255-RSA4096-BAB872.crt contains the ROOT CA.

-----BEGIN CERTIFICATE-----

```
MIIIFIDCCAwigAwIBAgIBATANBgkqhkiG9w0BAQwFADBBMRcwFQYDVQQLDA5QbHVn
IGFuZCBUcnVzdDEMMAoGA1UECgwDTlhQMRgwFgYDVQQDDA9OWFAgUm9vdENBd1I0
MDYwHhcNMtKwNDI1MTQzMzAzWhcNMzcwNDI1MTQzMzAzWjBBMRCwFQYDVQQLDA5Q
bHVnIGFuZCBUcnVzdDEMMAoGA1UECgwDTlhQMRgwFgYDVQQDDA9OWFAgUm9vdENB
d1I0MDYwggIiMA0GCSqGSIb3DQEBAQUAA4ICDwAwggIKAoICAQCwRBRevWxzoVln
bFOWxjhjFmX6hqPBB5o7pOVVGqvvdkSvcdZh7+hozTPzI7d5eCJWtiIYZ4xUXxiP8
MttIsssT4yrZjpSy2qTwbn5T/1B7a2e/A5JTGUCx05/uFATfZTs2pc0UydB68PnB
LVss8i1atpxN4TiY5NPchWM2mL7YwoKJxCNgZFs5WYxvm4OSOhMy7rAF010ujy1k
6L/XydcSv48G6ZlpCYH8Tn5cV7UEuNWpgq+0Wgpz6xz/ZHyfTxKhuBtj4eh3GyDv
Fmt1L1hJT/k17+Q19rW3U7OBV5w8ehuLoRNsNzG2zc9rFH1K9pVX9DRZQKI5XNo
1he7oSxJFkjsLn27w0fOHraOaeFrYuOrbKDA/X+cJzyKOvxWNhVyFRrynn2i058Wa
HuBqja7fnubexMP1Mr2b/hZV11SPmFVDk+B17VXpvC/tYXEqlSK1Tj/1gZ5tpmpm
SBBqcHsp0YQEkNLAnG+K9sFOKamf1Yt0IO/iyoYPUQ5IV644p0izt02myFrT04+k
Pwi+XwoRBVSK/Wri5vy16HzAJqPP1Nb6QKdOe1Q+NwNMyUySFzHavd3L4XaiNP4Z
iPgjSG/me8ozcPTFR29eibP17p9Y1hocmU4g2UURStVL0cpdOrc36BEv6DQ6RN4H
7N9VM9AIP+1047argR+ciYzky+Xd+wIDAQABoyMwITAPBgnVHRMBAf8EBTADAQH/
MA4GA1UdDwEB/wQEAWIBjANBgkqhkiG9w0BAQwFAAOCAgEAQsNB0/011ELP5rHN
rk9+sTh+Mr6Ye7geTYT1QRCCpLuyUuyc3051dSOKP3RorsNYD/aCJGhHDvh0ofyj
yKx51re2+b2foSVg+ZjQQf3qju+7bqh3tW1dD2b149+yRoCIplJZQ8V91ReH+33k
TzYNv/tvzYxnF8AF7wJ93zyafXJvIQ6AUv0Ddgtn203M853jxavjZHSVjTQXaR0
LwYCcNiE+bE2Z4owjerG3QAVG/ju8Xz+bhORy/+M1ib1F8KGZRzSA0E0ijMhhfZ
ZIP6eXjWWaoL+9iqD8F8/IXxW2zH37IzOTB6s2xcSFvjGbZVJ7CHy8/c4OQEoZpi
gujcPF5axOpI2RSnUPX+YFyu3od5URS3ybo9aT7b1CLURkjbhCGHCo04NL3h6Zg1
G2ktasOCL3moypWY0EYNwrHKyccVY3VVR1H89P03qQf5uUAADFcrpZ81B7yPy1mt
4qcZfbACzUt300UzUI1VDyHL6Nctw/1Da26aGKZMseHan6xEsdpiHMC5efv4Xd7U
k+bVedNwTKoodZn1K5WKTiZ3nUcSRAhmiaQrAb61KYEcDxRK5VPtr/MU6cL2PKZ
m9oZOhnfB5EbYJH4tWAite7j5kMrtolt+woeWP/dSmDg31ZCBmc3Sx0NaU12jQZ
2vnNPomcaE8BkvyrhEPkSw/VXTw=
-----END CERTIFICATE-----
```

10.9.1.1.2.2 Intermediate CA

The file IOT_NXP-01-CERT_IOT_4LAYER_CA_KEY-IoTInt4LayerCAvR406-01-20190425163534-RSA4096-540F19.crt contains the intermediate CA.

-----BEGIN CERTIFICATE-----

```
MIIFMjCCAqgAwIBAgIBAjANBgkqhkiG9w0BAQwFADBBMRcwFQYDVQQLDA5QbHVn
IGFuZCBUcnVzdDEMMAoGA1UECgwDTlhQMRgwFgYDVQQDDA9OWFAgUm9vdENBd1I0
MDYwHhcNMtKwNDI1MTQzNTA5WhcNMzcwNDI1MTQzNTA5WjBQMRCwFQYDVQQLDA5Q
bHVnIGFuZCBUcnVzdDEMMAoGA1UECgwDTlhQMScwJQYDVQQDB5OWFAgSW50ZXjt
ZWRpYXR1LTRMYX11ckNBd1I0MDYwggIiMA0GCSqGSIb3DQEBAQUAA4ICDwAwggIK
AoICAQCyji2V12sVG7PQNj8uzYSVA+cSCDvP/pCqvcA8ulaVuTfjLQVkJPeRUO
549JKWvE/318mW1mUApFcKOq1ud6/W9iFgKHdVpwKkOH1miIdHiOEHCu2Qr7EnKH
jMD5M/dhwqZdRxlojiMnEw7pt4x2gdndZQQFO5eHIBb/SIgeeUVreYnjRTfTRqqV
oZfYPV09IYx4Lm11FcCgODpn5TSIYvE8fGqGTm+j3+Q0g47N6Uve3SGdjL7aese
```

```
wGb6EPqGUqaSGLQgKy/L0K1gnVZXOeidFJjOe5QsKUQgWbCExcLObR2a3XzbIwXq
dq+ED1ovhrRDHH5VDRgK4+7WCFy6wQD9PPIy8WiOsCV2XCSixAM6XhWUYkLignp
30Yy5tzKhbCpyxe84ZM3sZG/PAojVAbwpVpj/m/px3N8jdQ/hE1eVgaoyTX0Pn9We
tXRQkUEBaluOm3Czaw1YN1E5LwpP8hYhWTMPVnf1z/R3rdswiWaA2+knWJCXkds
Kle8h4xNB0e0ZAzuleXh9JROsOqZUuh0+sGwPswl3p4KqauBzSG77tos1e4xp9AI
9P7JcnRsEeNjgTvbjBxIS1QdHW3FCnud7hM/OoTbGojwB3CHPIoIjmWCW7ci6beD
DUwOo2CfjZ+F1QErar2XarQrPLdSFTd+1JJqUy+tiNoWPJafYQIDAQABoyYwJDAS
BgNVHRMBAf8ECDAGAQH/AgEBMA4GA1UdDwEB/wQEAWIBBjANBgkqhkiG9w0BAQwF
AAOCAgEAW0JtZaJCEbm4+DmYa32WNvqjjxgdmRHjkJtfv+wkt0Kz71MHG0qSQBHH
UkVOKBFm/wjF+1H+/7X9ZdiFsO3b9P2JXHg1PXgU9NvtuYZBAsiy0VRxVSukp+JH
7biOFNx5nwerYv9C1JG2VRPr+JzDUP1/QV9XCftQVnpym/pETLghOwaNq4Yjzg+
MxLw4sWsJL5DZrG2/jViHogoCnXz1rFVWF1BBvEbKnthYUxqL+Xc4vWCYLvMHC
LbZMGef05HrcPwtAS13y8qpat5MGRXDjgFqGUyoaPoxRya4d0exjw369aV/YAIsx
wgtAiWrvL4UPOUGopCMac4GEbpPA60XHmSVXYR+CCKyLwLD7rRR8Vpk5cUM2jvxI
5NMP17Wa+QD4jp8sWsSebI83+qL/0TdWgRbD/FW2igrR0ird8wgQcIhtCrJcdYMT
MjRtJc4m/iJ+OH2KaryxQi8oEkB3onsIZXvSNgXvTp4SkLw9pPheEwSKVvbo1DoF
QB4ek6ynjWGgoaH15tx2i+tRH+mEP7+Ccw32Hdx+xo8BYvMaBXJ18M5rem7k4PsQ
tLNjn9CN0FPT8j2ksk8jMi4s/c9EHY5NYNgjTG6MQ3eSS1QaoVYPi4NCDe2KWP2k
ooGZMUL1CfXbP86zQz5xBJvd/+Rmdn+mvLoabzmFVPS9E1yH6fE=
-----END CERTIFICATE-----
```

Note:

The certificates shown here were last Updated on June 18, 2019 in this page/document.

10.9.1.2 Certificate Chains : DEV Kit

Certificates for the DEV Kit / Ease Of Use Configuration.

- [Certificate Chain for SE050](#)
 - [Intermediate CA : ECC](#)
 - [Intermediate CA : RSA](#)

10.9.1.2.1 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](#). More information on these certificates can be found in the application note on SE050 configurations: [AN12436](#)

10.9.1.2.1.1 Intermediate CA : ECC

The file `CloudConn-Intermediate-ECC_OEF_A1F4.crt` is certificate used for Dev Kit configuration.

```
-----BEGIN CERTIFICATE-----
MIIBujCCA WGgAwIBAgIKBABQAQAEofQAADAKBggqhkJOPQQDAjBQMRCwFQYDVQQL
DA5QbHVnIGFuZCBUcnVzdDEMMAoGA1UECgwDTlhQMScwJQYDVQQDB5OWFAgSW50
ZXJtZWRpYXR1LTRMYX11ckNbdkUyMDUwHhcNMtkwNjE4MDAwMDAwWhcNMzQwNjE0
MDAwMDAwWjBdMRcwFQYDVQQLDa5QbHVnIGFuZCBUcnVzdDEMMAoGA1UECgwDTlhQ
MTQwMgYDVQQDDCtDbG91ZENvbmtSW50ZXJtZWRpYXR1LTA0MDA1MDAxMDAwNEEx
RjQtRUNDMFkwEwYHKoZ1zj0CAQYIKoZ1zj0DAQcDQgAENi7KN2Z6tbFuhjRrqgHb
xzUWSCUrldPmFP8xpsIOEHM8WJPvj8D+MCeu6Y2WU/ILhxplY3TJ2hGZtV4foO3pq
NqMWMBQwEgYDVR0TAQH/BAgwBgEB/wIBADAKBggqhkJOPQQDAgNHADBEAiB0waP1
4dcmbqBLufQRscoPVzSeqT2dfdwg0W0g/FlaPgIgS2ppnaRASzc5rCOeV5AeMb2
j7IKEHDyeMRP0wlaptU=
-----END CERTIFICATE-----
```

10.9.1.2.1.2 Intermediate CA : RSA

The file CloudConn-Intermediate-RSA_OEF_A1F4.crt is certificate used for Dev Kit configuration.

```
-----BEGIN CERTIFICATE-----
MIIFRzCCAY+gAwIBAgIKBABQAQAEofQAADANBgkqhkiG9w0BAQsFADBQMRcwFQYD
VQQLDA5QbHVnIGFuZCBUcnVzdDEMMAoGA1UECgwDTlhQMScwJQYDVQQDDB50WFAG
SW50ZXJtZWRpYXR1LTrMYX11ckNBd1I0MDYwHhcNMTkwNjE4MDAwMDAwWhcNMzQw
NjE0MDAwMDAwWjBdMRCwFQYDVQQDCTDbG91ZENvb4tSW50ZXJtZWRpYXR1LTA0MDA1MDAxMDAw
NEExRjQtUlNBMIICIJAnBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAzkNa+y5n
9CQK9Aw4F1Lrm17chE3CBFgnp49OSNII0mI2G/YmhI2zotydgIp17Wk+gJd1wpQM
rKuh2RmLihoIyLmuuOVxKbn1Tw35tshij48linj7G6n4x7SmU2khjPqsjdF4ncp
VK6IcXoNQ+zteN8/q5Pcbmozo+UriUhLGdu6By/Z3+BseTsNYJhouIzN8CwZ1h+r
LRgjpiodrwHE0jVbe74Fqjcb7TEqACHZNEaCH1fnSs4BaiiMOSMGGhGrPMHxb0z
VHESSJGZ3aa7xzFakoP6hNz6m82q14WYMM/2fbuOh2mIgrEexz2zTGsG1a3+wt
uy46QVCivEMwsMDYzOjocO1CbAsRUfT9pbzdKVxt5qAPpIQLxOzSkgDuOw5N/jdN
Kmt/sa7gh8uRqnQVvzjKsmDg0TqolPuTrt2tS371Mw1yKToAeo09w/HkMb6CF7GB
EgTTnob1MGUpJRdk9MM/PMi15tsXT30G1MVCxORDk6y9nBsxLAPzBC3SQajmk6T1
KWPgBlrjNIFmkosCKQPPzn1cdygpbCKHeRt29X+TsQq12tDTHMUTdg5NWJ/6gPEk
2ur12J2iEF9LaAu3dM5aKfLauoyvKeN1Z6FUBVQz9PgLRrcCR52eiUn6AOR9gZZN
Lozp/uQUU+CBMw0daEBI7+xethA9F426CCMCAwEAAaMWMBQwEgYDVR0TAQH/BAgw
BgEB/wIBADANBgkqhkiG9w0BAQsFAAOCAgEAJLw+RCn/qhLPn i7NwkTfi69bnrF
je6nhhgvleZ71xUPu3Jdd+swgfdZmKDnUfr4mQJ21/rj2Z4yx7WA7Cy3IjG52d
9A9/jrqUN8T9AmqdME9e4rwTmjzCtKx91yFex+jcisJphcc6zkoDmp/XaumXi9GW
M+J0laY+hj6nw9BWbd/wGDxe2mzBjdWNtP8jb3CKuqJQexWB5pjNUnexdJSDFXs
Hf0NYtolzK7dh/+GMUfdTCX/8g3hN2Won1DD6/fTXdgCSXExaBz0EhsethdFb3y6
vR+MvngMoZhbyGH5VzQ2sESo9vGzfPSiM8mHvjaV++hDduVEpodoxSE/C8TdNpUZ7
Lh6E1JYcUbDE2VV93/dYaiM5Cag/SuKG/8phZ3g9eXT9oPmOEtKXg9bdMJoNwJ0N
LiAW3eFdBBmDGB/6D5vSOD3YkPhtBUIC+/9vBL0KNx9V9/4xWZAnmTtjLfYLoyst
IE5Qnr0TpqZ123+Tvj9X7ulNUuu7MMHtiAs2SKM9iWMXAd/UW6Vq5x7SKjQ7b6FQ
vCn/CAZn9uV4ixwDzbIoQV+4ps9QuyqVGJbtqEctxYo896ulTeJAFyiPnfeF1WdM
gPmab4mVjTyikESCtSkM2nNYVkJQ5ANhfEQ8XVoRsdJmp3rMl6MD1n1zjfSyEHrQ
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).

10.9.2 SE051 Certificate Chains

10.9.2.1 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 SE051. More information on these certificates can be found in the application note on SE051 configurations: [AN12973](#)

10.9.2.1.1 ECC

This directory contains the ECC chain of trust for cloud on-boarding.

10.9.2.1.1.1 ROOT CA

The file RootCAvE305.crt contains ROOT CA.

```
-----BEGIN CERTIFICATE-----  
MIIB1jCCAVmgAwIBAgIBATAMBggqhkjOPQQDAwUAMEExFzAVBqNVBAAsMD1BsdWcg  
YW5kIFRydXN0MQwwCgYDVQQKDANOWFAXGDAWBqNVBAMMD05YUCBSb290Q0F2RTMw  
NTAeFw0xOTAzMjAxNTI2MDRaFw0zNzAzMjAxNTI2MDRaMEExFzAVBqNVBAAsMD1Bs  
dWcgYW5kIFRydXN0MQwwCgYDVQQKDANOWFAXGDAWBqNVBAMMD05YUCBSb290Q0F2  
RTMwNTB2MBAGByqGSM49AgEBGSuBAAiA2IABNRrkWrw7iwM3oTw1Ay8I1yzOF+g  
OTNFzqfn/93N1xcK8kNEA8wNuuvjzsDXKHx7O1jrzGZfy7YLjKYTwZR5wURXrE7lp  
PIwwdWE3OokTmhiMiFXnzSgSHCgtb+VF6nv76aMjMCEwDwYDVR0TAQH/BAUwAwEB  
/zAOBgNVHQ8BAf8EBAMCAQYwDAYIKoZIzj0EAwMFAANpADBmAjEA7cCFz6PcBHki  
HRtbE3qi0Lj9iqxe8/2w8XrLc1AYhpMzzOQC05j3ZmgJ22B1bLk3AjEAhVnhuawO  
Zh1Jqwf7zySh/rNJezeFhGTyHAjQ9tTfy9eZAdc25feEN4j/Ad+7TF1Rg  
-----END CERTIFICATE-----
```

10.9.2.1.1.2 Intermediate CA

The file Intermediate-4LayerCAvE205.crt contains the Intermediate CA.

```
-----BEGIN CERTIFICATE-----  
MIIBYjCCAU6gAwIBAgIBBDAMBggqhkjOPQQDAgUAMEExFzAVBqNVBAAsMD1BsdWcg  
YW5kIFRydXN0MQwwCgYDVQQKDANOWFAXGDAWBqNVBAMMD05YUCBSb290Q0F2RTMw  
NTAeFw0xOTAzMjAxNTQyNTRaFw0zNDAzMjAxNTQyNTRaMFAxFzAVBqNVBAAsMD1Bs  
dWcgYW5kIFRydXN0MQwwCgYDVQQKDANOWFAXJzA1BgNVBAMMHk5YUCBJbnR1cm11  
ZG1hdGUtNExheWVYQ0F2RTIwNTBZMBMGBqGSM49AgEGCCqGSM49AwEHA0IABo+d  
HK3Oa4FnkKN9/1JQpR2KarpwmwNRaEG6Zb+kzTwnXRw4Cvknd8IAcjXHqb93Vfx5  
rO+4MjX/gzqYagJByWejJjAkMBIGA1UdEwEB/wQIMAYBAf8CAQEwDgYDVR0PAQH/  
BAQDAgEGMAwGCCqGSM49BAMCBQADaAAwZQIxAPvXuv1W+zksBBz0NyyzpgFP1rCj  
IZOHpfZhaERw/DEj+4aAESa5vgtiR3CspIP5pAIwRsD1Z9fdBnPS1aVMmNAXjA1Z  
FPhbV7A0OXdSYhI8M5uTENrdezvsYYr2jfqIEcp  
-----END CERTIFICATE-----
```

10.9.2.1.2 RSA

This directory contains the RSA chain of trust for cloud on-boarding.

10.9.2.1.2.1 ROOT CA

The file RootCAvR406.crt contains the ROOT CA.

```
-----BEGIN CERTIFICATE-----  
MIIFIDCCAwigAwIBAgIBATANBgkqhkiG9w0BAQwFADBBMRCwFQYDVQQLDA5QbHVn  
IGFuZCBUcnVzdDEMMAoGA1UECgwDTlhQMRgwFgYDVQQDDA90WFAgUm9vdENBd1I0  
MDYwHhcNMTkwNDI1MTQzMzAzWhcNMzczwNDI1MTQzMzAzWjBBMRCwFQYDVQQLDA5Q  
bHVnIGFuZCBUcnVzdDEMMAoGA1UECgwDTlhQMRgwFgYDVQQDDA90WFAgUm9vdENB  
d1I0MDYwggIiMA0GCSqGSIb3DQEBAQUAA4ICDwAwggIKAoICAQCwRBRveWxzoVln  
bFOWxhjFmX6hqPBB5o7pOVVGqvvdksVcdZh7+hozTPzI7d5eCJWtiIYZ4xUXxiP8  
MttIsssT4yrZjpSy2qTwbn5T/1B7a2e/A5JTGUCx05/uFATfZTs2pcouydB68PnB  
LVsS8i1atpxN4TiY5NPchWM2mL7YwoKJxCNgZFss5WYxvm4OSOhMy7rAF010ujy1k  
6L/XydcSv48G6ZlpCYH8Tn5cv7UEuNWpgq+0Wgpz6xz/ZHyfTxKhuBtj4eh3GyDv
```

```
Fmt1L1hJT/k17+Q19rW3U7OBV5w8ehuLoRNsNzG2zc9rFH1K9pVX9DRZQKI5XNoH
1he7oSxJFkjsLn27w0fOHraOaeFrYuOrbKDA/X+cJzyKOvxWNhVyFRryN2i058Wa
HuBqja7fnubexMP1Mr2b/hZV11SPmFVDk+B17VXpvC/tYXEqlSK1Tj/1gZ5tpmpm
SBBqchsp0YQEkNLAnG+K9sFOKmf1Yt0IO/iyoYPUQ5IV644p0iztO2myFrT04+K
Pwi+XwoRBVSK/WrI5vy16HzAJqPP1Nb6QKdOe1Q+NwNMyUySFzHavd3L4XaiNP4Z
iPgjSG/me8ozcPTFR29eibPl7p9YlhocmU4g2UURStVL0cpd0rc36BEv6DQ6RN4H
7N9VM9AIP+1o47argR+ciYzkY+Xd+wIDAQABoyMwITAPBgNVHRMBAf8EBTADAQH/
MA4GA1UdDwEB/wQEAWIBBjANBgkqhkiG9w0BAQwFAAOCAgEAQsNB0/011ELP5rHN
rk9+sTh+Mr6Ye7geTYT1QRCCpLuyUuyC3051dSOKP3r0rsNYD/aCJGhHDvh0ofyj
yKx5lre2+b2foSVg+ZjQQf3qju+7bqh3tW1dD2bI49+yRoCIplJZQ8V91ReH+33k
TzYNv/tvzYxnF8AF7wJ93zyafXJvIQ6AUv0DdgtN203M853jxavjZHSVjTQXaR0
LwYCcNiE+bE2Z4owjerG3QAVG/ju8Xz+bh0FRy/+M1ib1F8KGZRzSA0E0ijMhfZ
ZIP6eXjWWaoL+9iqD8F8/IXxW2zH37IzOTB6s2xcSFvjGbZVJ7CHy8/c4OQEoZpi
gujcPF5axOpI2RSnUPX+YFyu3od5URS3ybo9aT7b1CLURkjdhcGHCo04NL3h6Zg1
G2ktasOCL3moypWY0EYNwrHKyccVY3VVR1H89P03qQf5uUAADFcrpZ81B7yPylmt
4qcZfbACzUt300UzUI1VDyHL6NcTw/1Da26aGKZMseHan6xESDpiHMC5efv4Xd7U
k+bVedNwTKooDzn1K5WKTiZ3nUcSRAhdmiaQrAb61KYEcDxRK5VPtr/MU6cL2PKZ
m9oZ0hnfB5EbYJH4tWAite7j5kMrt0ltT+woeWP/dSmDg31ZCBmc3Sx0NaU12jQZ
2vnNPomcaE8BkvyrhEPkSw/VXTw=
-----END CERTIFICATE-----
```

10.9.2.1.2.2 Intermediate CA

The file `Intermediate-4LayerCAvR406.crt` contains the intermediate CA.

```
-----BEGIN CERTIFICATE-----
MIIFMjCCAxqgAwIBAgIBAjANBgkqhkiG9w0BAQwFADBBMRcwFQYDVQQLDA5QbHVn
IGFuZCBUcnVzdDEMMAoGA1UECgwDTlhQMRgwFgYDVQQDDA90WFAGUm9vdENBd1I0
MDYwHhcNMtKwNDI1MTQzNTA5WhcNMzQwNDI1MTQzNTA5WjBQMRcwFQYDVQQLDA5Q
bHVnIGFuZCBUcnVzdDEMMAoGA1UECgwDTlhQMScwJQYDVQQDDB50WFAGSw50ZXjt
ZWRpYXR1LLTRMY11ckNBd1I0MDYwggIiMA0GCSqGSIb3DQEBAQUAA4ICDwAwggIK
AoICAQCyji2V12sVG7PQNj8uzYSVA+cSCDvP/pCqvca8ulaVuTfjLQVkJLQVkJLQVkJ
549JKWvE/318mW1mUApFcK0q1ud6/W9iFgKHdVpwKkOHlmiIdHiOEHCu2Qr7EnKH
jMD5M/dhwqZdRxlojimnEw7pt4x2gdndZQZFO5eHIBb/SIgeeUVreYnjRTfTRqqV
oZfYPV09IYx4Lm11FcCgODpn5TSIYvE8fGqGTM+j3+Q0g47N6Uve3SGdjL7aese
wGb6EPqGUqaSGL0gKy/L0K1gnVZX0eidFJj0e5QsKUQgWbCExcLobR2a3XzbIwXq
dq+ED1ovhrRDHH5VDRgK4+7WCfy6wQD9PPIy8Wi0xsCV2XCSixAM6XhWUYkLignp
30Yy5tzKhbCpyxe84ZM3sZG/PAojVAbwpVpj/m/px3N8jdQ/hE1eVgaoYTx0Pn9We
tXRQkUEBluOm3Czaw1YN1E5Lwp8hYhWTMPVnf1Z/R3rdswiWaA2+knWJCXkdsk
Kle8h4xNB0e0ZAzuleXh9JROs0qZUuh0+sGwPsw13p4KqauBzSG77tos1e4xp9AI
9P7JcnRsEeNjgTvbjBxIS1QdHW3FCnud7hM/OoTbGojwb3CHPIOIJmWCW7ci6beD
DUwOo2CfjZ+F1QErar2XarQrPLdSFTd+1JjQyU+tiNoWPJa fYQIDAQABoyYwJDAS
BgNVHRMBAf8ECDAGAQH/AgEBMA4GA1UdDwEB/wQEAWIBBjANBgkqhkiG9w0BAQwF
AAOCAgEAW0JtzaJCEbm4+DmYa32WnvqjjxgdmRHjkJtfv+wkt0Kz71MHG0qSQBHH
UkVokBFm/wjF+1H+/7X9ZdiFs03b9P2JXhg1PXgU9NvtuYZBAsiy0VRxVSukp+jH
7biOFNx5nwerYv9C1JG2VRPr+jzDUP1/QV9XCftQVnpym/pETLghOwaNq4Yjzg+
MxLw4sWsJL5DzRg2/jViHhogoCnXz1rFVWF11BbvEbKnthYUxqL+x4vWCYLvMHC
LbZMGef05HrcPwtAS13y8qpat5MGRXDjgFqGUyoaPoxRya4d0exjw369aV/YAIsx
wgtAiWrvL4UPOUGopCMac4GEbpPA60XHmSVxyR+CCKyLwLD7rRR8Vpk5cUM2jvxI
5NMP17Wa+QD4jp8sWsSebI83+qL/0TdWgRbD/FW2igrR0ird8wgQcIhtCrJcdYMT
MjRtJc4m/iJ+OH2KaryxQi8oEkB3onsIZXvSNgXvTp4SkLw9pPheEwSKVvb01DoF
QB4ek6ynjWGgoaH15tx2i+tRH+mEP7+Ccw32HdX+xo8BYvMaBXJI8M5rem7k4PsQ
tLNjn9CN0FPT8j2ksk8jMi4s/c9EHY5NYNgjTG6MQ3eSS1QaoVYPi4NCDe2KWP2k
ooGZMu1ICfXbP86zQz5xBJvd/+Rmdn+mvLoabzmFVPS9E1yH6fE=
-----END CERTIFICATE-----
```

Note:

The certificates shown here were last Updated on October 16, 2020 in this page/document.

10.9.2.2 Certificate Chains : DEV Kit

Certificates for the DEV Kit / Ease Of Use Configuration.

- [Certificate Chain for SE051](#)
 - [Intermediate CA : ECC](#)
 - [Intermediate CA : RSA](#)

10.9.2.2.1 Certificate Chain for SE051

The directory `demos/Certificate_Chains/SE051/0001_A201` contains Intermediate Certificates used in Dev Kit configurations. It is signed by [Certificate Chains : ROOT](#). More information on these certificates can be found in the application note on SE050 configurations: [AN12973](#)

10.9.2.2.1.1 Intermediate CA : ECC

The file `CloudConn-Intermediate-ECC_OEF_A201.crt` is certificate used for Dev Kit configuration.

```
-----BEGIN CERTIFICATE-----
MIIBujCCAWGgAwIBAgIKBABQAQABogEAADAKBggqhkJOPQQDAjBQMRCwFQYDVQQL
DA5QbHVnIGFuZCBUcnVzdDEMMAoGA1UECgwDTlhQMScwJQYDVQQDDB50WFAGSW50
ZXJtZWRpYXR1LTRMYX11ckNBdkUyMDUwHhcNMkNTE1MDAwMDAwWhcNMzQwNTEX
MDAwMDAwWjBdMRCwFQYDVQQLDA5QbHVnIGFuZCBUcnVzdDEMMAoGA1UECgwDTlhQ
MTQwMgYDVQQDDCtDbG91ZENvbmtSW50ZXJtZWRpYXR1LTA0MDA1MDAxMDAwMUEy
MDEtRUNDMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEeEU5NJXlvO0NY8kquO
VZluA+pQBPMxONWqVqxob/wXXjwxfcG7Jxla4yACIyyAR8Um6yhkDcK4T7gtf28r
HKMWMQwEgYDVR0TAQH/BAgwBgEB/wIBADAKBggqhkJOPQQDAgNHADBEAiAxmSyR
388EKFo40juspNEhhLBfvlsa5hpEJTxi/xGQIgL9mWld18LvnA9qutAmBE7UnA
2tc5GoX30IwBQEAWTo=
-----END CERTIFICATE-----
```

10.9.2.2.1.2 Intermediate CA : RSA

The file `CloudConn-Intermediate-RSA_OEF_A201.crt` is certificate used for Dev Kit configuration.

```
-----BEGIN CERTIFICATE-----
MIIFRzCCAY+gAwIBAgIKBABQAQABogEAADANBgkqhkiG9w0BAQsFADBQMRcwFQYD
VQQLDA5QbHVnIGFuZCBUcnVzdDEMMAoGA1UECgwDTlhQMScwJQYDVQQDDB50WFAg
SW50ZXJtZWRpYXR1LTRMYX11ckNBd1I0MDYwHhcNMkNTE1MDAwMDAwWhcNMzQw
NTEXMDAwMDAwWjBdMRCwFQYDVQQLDA5QbHVnIGFuZCBUcnVzdDEMMAoGA1UECgwD
TlhQMTQwMgYDVQQDDCtDbG91ZENvbmtSW50ZXJtZWRpYXR1LTA0MDA1MDAxMDAw
MUEyMDEtUlNBMIICjANBgkqhkiG9w0BAQEFAOCAg8AMICCgKCAGEA0guMQQvD
pdzjTU/zB4S4BWwhIKoyZvhCyEA9eaHnWgaWmUwGPnM8P5M0F1vPBtaW/Xq8kJGe
K7cAPy2VKrNCFVg00qczL7iHCnmgmaHM0IGim51KoP0pRcJy206PW+uRvozUkYEK
jZHLnsPdVhsP5+YGmU5WWYu4jnNkzFpJtR3fBJyazav0/lr0kLvSg6faH3nrqRGJ
thsCB1szKPQ/Ogi/Ew6P3J7pfoBTXngon3uz2hPgfvnkau7zqCEue8OG2+CyfKKB
N8aiEce3FM4KHSi4wnE93gYzq+uf7PBi7pSQXV4uuop54rlZRZ2AeB4yViEn9Icf
ZRBrqKtg+b3fRnK4dSqbMjHb30wCgRGWHMn/+WdujZi9hdCxhflyQIVB6Pa8RTxn
kwMrYqWX+KQyhVKY1dazV5mxqwxE3pArroObq3tCLMLEpaVjG/zSXD/sBHqKDyuG
qgymsmqCdiCgFBvaOYdT30ccGHFXmltzXWvgfCGxvw2neMTfacYtGbMh1Oh409jX
91i2CGB5ZNNHBdcH1hpbygWiyp60qyMmEO2yBH1Age/kBgbWMppA8W3cC+xP+vW
94RiN+uiuRBr21iUg4PSizNLrZLW8c90HcSt9p0pwvvBxt67inu/Yh52AUz2/m61
2uhzgRDpR6TvhYhFg2RnmqiSQPtVYUkEr1ccAwEAAaMWMBQwEgYDVR0TAQH/BAgw
BgEB/wIBADANBgkqhkiG9w0BAQsFAAOCAgEAYvgJL4TAU7sWnydj1j20FHMdcG/8
z2sohJykn8W+nayxeCtdtil+kMxTDPGzER2rCmHfWPrLg95rDYIwWscDnuPSg2za
W8RzwyEwjv27AYZpFB57pPGw25RT20cMPTIpZ01F3aeKEguy3SWSU/Zs6Hgj/8IS
6iJfr0VnB/znthaZfd0ECZknx6bZrbyL09Ls0y/7ygxytYzDrC30vSVaaP/NWZFCp
```

```
qmH1paGtSYr1LtPFYHS7YfLmmYfgHk5bBpzYKCRlGov8j3b35xvj/NSETWnnojfA  
xg4PwdAzKBTn8TLjIkklwcvkdwHYFUIvqcavN4f4YgYKJ48MWwKvMQBF1X73Q1C  
E6dXS1TqwULX0staIS4yl9c1gVLkdCx1RObKPq222IfG1OzQaBXt5iIFWA4d4PZS  
dneKwDYrH9k/1UqlWMO/q87m8Tcuvo85kML8sshPWgzs2OoA30bEaOXg1D1WNhnO  
t+NzdjbXc9MQPzr0r6Gk7BxL5W4wGArujMk1jk781KpeTh5R4IFENI31Ol0PShOa  
0QnL3DJbpdVMUu2LuE8FsV3k+UdHuboego1hpP59zpN6goUoBMfGhMdI2wb16wv  
hKFpiDiBq+bu1WIBOA2DD5kUlTid7yqTp5NtI2TYvXzb5NSehqsCaHopz4Q/+j0w  
tJ8ao7HfcVhb7Yc=  
-----END CERTIFICATE-----
```

Note:

The certificates shown here were last Updated on October 16, 2020 in this page/document. These certificate chains are only for Dev Kit configuration (OEF A201).

10.10 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:

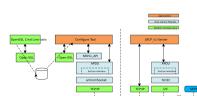
The JRCP_v1 server was previously known as RJCT server.

10.10.1 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=I2C protocol.

10.10.2 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:~# ./jrccpv1_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 SMCOM 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
```

10.11 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 10.6.9.2.1 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 35C256458958A34F6136155F8209D6CD # Trailing comment  
MAC AF177D5DBDF7C0D5C10A05B9F1607F78 # Optional trailing comment  
DEK A1BC8438BF77935B361A4425FE79FA29 # Optional trailing comment
```

Replace the ENC, MAC and DEK keys with your own keys.

For information on rotating Platform SCP03 keys, refer to [Section 5.7.6 SE05X Rotate PlatformSCP Keys Demo](#)

10.12 Write APDU to buffer

See `write-apdu-to-buf`/`write-apdu-to-buf`.

10.13 Class Hierarchy

- [Struct_SE05x_I2CM_cmd](#)
- [Struct_SE_AuthCtx](#)
- [Struct_sems_lite_available_mem_t](#)
- [Struct_sss_se05x_object](#)
- [Struct_sss_se05x_session](#)
- [Struct_sss_se05x_tunnel_context](#)
- [Struct_sub_component_metaData_t](#)
- [Struct_datastore_stream_context_t](#)
- [Struct_handle_request_payload_args_t](#)
- [Struct_iot_agent_datastore_interface_t](#)
- [Struct_iot_agent_datastore_t](#)
- [Struct_iot_agent_dispatcher_context_t](#)
- [Struct_iot_agent_endpoint_cache_table_entry_t](#)

- [Struct iot_agent_keystore_interface_t](#)
- [Struct iot_agent_keystore_key_ref_t](#)
- [Struct iot_agent_keystore_t](#)
- [Struct multicast_package_t](#)
- [Struct NXECKey03_StaticCtx_t](#)
- [Struct NXSCP03_AuthCtx_t](#)
- [Struct NXSCP03_DynCtx_t](#)
- [Struct NXSCP03_StaticCtx_t](#)
- [Struct SE05x_Applet_Feature_Disable_t](#)
- [Struct SE05x_Applet_Feature_t](#)
- [Struct Se05x_AppletFeatures_t](#)
- [Struct SE05x_AuthCtx_ECKey_t](#)
- [Struct SE05x_AuthCtx_ID_t](#)
- [Struct SE05x_ExtendedFeatures_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 SE05x_TimeStamp_t](#)
- [Struct Se05xApdu_t](#)
- [Struct Se05xPolicy_t](#)
- [Struct Se05xSession](#)
- [Struct SE_Connect_Ctx_t](#)
- [Struct sems_lite_agent_ctx_t](#)
- [Struct sems_lite_agent_request_ctx_t](#)
- [Struct sems_lite_SEAppInfoList_t](#)
- [Struct service_link_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_desfire_changekey_authId_value_u](#)
- [Struct sss_policy_file_u](#)
- [Struct sss_policy_internal_sign_tbs_value_u](#)
- [Struct sss_policy_key_drv_master_keyid_value_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_rng_context_t](#)
- [Struct sss_se05x_symmetric_t](#)
- [Struct sss_session_t](#)
- [Struct sss_symmetric_t](#)
- [Struct sss_tunnel_t](#)
- [Struct tlvHeader_t](#)
- [Enum iot_agent_stream_type_t](#)
- [Enum sems_lite_recovery_status_t](#)
- [Enum sems_lite_status](#)
- [Enum sems_lite_teardown_status_t](#)
- [Enum sems_lite_upgradeProgress_status_t](#)
- [Enum sems_lite_version_check_result_t](#)
- [Enum eSE05xType_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_ECDHAlgo_t](#)
- [Enum SE05x_ECPMAlgo_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_MoreIndicator_t](#)
- [Enum SE05x-Origin_t](#)
- [Enum SE05x_P1_t](#)
- [Enum SE05x_P2_t](#)
- [Enum SE05x_PAKEMode_t](#)
- [Enum SE05x_PAKEState_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_SPAKE2PlusDeviceType_t](#)
- [Enum SE05x_SW12_t](#)
- [Enum SE05x_SymmKeyType_t](#)
- [Enum SE05x_T4T_Access_Ctrl_t](#)
- [Enum SE05x_T4T_INS_t](#)
- [Enum SE05x_T4T_Interface_Const_t](#)
- [Enum SE05x_T4T_Operation_Const_t](#)
- [Enum SE05x_T4T_Read_Ctr_Operation_t](#)
- [Enum SE05x_TAG_t](#)
- [Enum SE05x_TLSPerformPRFType_t](#)
- [Enum SE05x_TransientIndicator_t](#)
- [Enum SE05x_TransientType_t](#)
- [Enum SE_AuthType_t](#)
- [Enum smStatus_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_a8_t](#)
- [Enum sss_mode_t](#)
- [Enum sss_policy_type_u](#)
- [Enum sss_s05x_sesion_prop_a8_t](#)
- [Enum sss_s05x_sesion_prop_u32_t](#)
- [Enum sss_session_prop_a8_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](#)

10.14 File Hierarchy

- [Directory doc](#)
 - [Directory input](#)
 - [File modules.h](#)
 - [File sss_object.h](#)
- [Directory hostlib](#)
 - [Directory hostLib](#)
 - [Directory inc](#)
 - [File modules.h](#)
 - [File nxEnsure.h](#)
 - [File nxScp03_Types.h](#)
 - [File se05x_enums.h](#)
 - [File se05x_tlv.h](#)
 - [Directory se05x_03_xx_xx](#)
 - [File se05x_04_xx_APDU_apis.h](#)
 - [File se05x_APDU.h](#)
 - [File se05x_APDU_apis.h](#)
- [Directory nxp_iot_agent](#)
 - [Directory inc](#)
 - [File nxp_iot_agent.h](#)
 - [File nxp_iot_agent_datastore.h](#)
 - [File nxp_iot_agent_dispatcher.h](#)
 - [File nxp_iot_agent_keystore.h](#)
 - [File nxp_iot_agent_service.h](#)
 - [File nxp_iot_agent_utils.h](#)
- [Directory semslite](#)
 - [Directory doc](#)
 - [File modules.h](#)
 - [Directory lib](#)
 - [Directory inc](#)
 - [File sems_lite_agent_common.h](#)

- [File sems_lite_agent_context.h](#)
- [File sems_lite_api.h](#)
- [File sems_lite_api_ver.h](#)
- [Directory sss](#)
 - [Directory doc](#)
 - [File modules.h](#)
 - [Directory ex](#)
 - [Directory inc](#)
 - [File ex_sss_main_inc.h](#)
 - [Directory inc](#)
 - [File fsl_sss_api.h](#)
 - [File fsl_sss_policy.h](#)
 - [File fsl_sss_se05x_apis.h](#)
 - [File fsl_sss_se05x_types.h](#)
 - [Directory plugin](#)
 - [Directory mbedtls](#)
 - [File sss_mbedtls.h](#)

10.15 Full API

10.15.1 Classes and Structs

10.15.1.1 Struct _SE05x_I2CM_cmd

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.1.1.1 Struct Documentation

struct _SE05x_I2CM_cmd

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](#)

10.15.1.2 Struct _SE_AuthCtx

- Defined in [File nxScp03_Types.h](#)

10.15.1.2.1 Struct Documentation

struct _SE_AuthCtx

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::\[anonymous\]\] 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::\[anonymous\]::\[anonymous\]\] extension](#)

Reserved memory for implementation specific extension

[SE05x_AuthCtx_ID_t idobj](#)

For UserID/PIN based based Authentication

[NXSCP03_AuthCtx_t scp03](#)

For PlatformSCP / Applet SCP.

Same SCP context will be used for platform and applet scp03

10.15.1.3 Struct _sems_lite_available_mem_t

- Defined in [File sems_lite_api.h](#)

10.15.1.3.1 Struct Documentation

[struct _sems_lite_available_mem_t](#)

Represents available memory for different components , such as COD,COR PersistentMemory, IDX

Public Members

uint32_t availableCODMemory

Indicates teh available memory for COD (Card Operating System)

uint32_t availableCORMemory

Indicates teh available memory for COR (Card operating Runtime)

uint32_t availableIDX

Indicates Available IDX

uint32_t availablePersistentMemory

Indicates Available PersistentMemory

uint32_t freePHeapCentralGap

Indicates amount of free memory in the central gap of the PHeap

uint32_t freeTransient

Represents free transient memory from cOD from JCOP

10.15.1.4 Struct _sss_se05x_object

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.1.4.1 Struct Documentation

`struct _sss_se05x_object`

An object (secure / non-secure) within a Key Store.

Public Members

`uint32_t cipherType`

cipherType type from [sss_cipher_type_t](#)

`SE05x_ECCurve_t curve_id`

If this is an ECC Key, the Curve ID of the key

`uint8_t isPersistant`

Whether this is a persistant 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](#)

10.15.1.5 Struct _sss_se05x_session

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.1.5.1 Struct Documentation

`struct _sss_se05x_session`

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.

10.15.1.6 Struct _sss_se05x_tunnel_context

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.1.6.1 Struct Documentation

`struct _sss_se05x_tunnel_context`

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 terminates to

10.15.1.7 Struct _sub_component_metaData_t

- Defined in [File sems_lite_api.h](#)

10.15.1.7.1 Struct Documentation

`struct _sub_component_metaData_t`

Public Members

`size_t aidLen`

Length of aid

`uint8_t minimumPreviousVersion[2]`

Minimum version number of this subcomponent as installed on the secure element before this script is executed. If this field is omitted there is no minimum version requirement, e.g. initial Installation of an applet. If the minimum PreviousVersion is 4.11 but the secure element contains version 3.2, the intermediate update packages need to be applied to reach version 4.11 before this package. Version information major.minor, according to java card virtual machine package versions, both values in range 0 to 127. byte 0 is Major value, byte 1 is Minor value.

`size_t nameLen`

Length of name

`const uint8_t *pAid`

The Application Identifier (AID) of the Executable Load File (ELF) which is in this subcomponent. This is stored as string to have it formatted in hexadecimal and therefore recognizable form. Encoded as binary.

`char *pName`

A human readable name for this subcomponent.

`struct _sub_component_metaData_t *pNextSubComponentMetaData`

Next subcomponent meta data

`char *pSWContentRegister`

NXP Software Content Register, specifies the Outgoing License, mostly: LA_OPT_NXP_Software_License.

`size_t swContentRegisterLen`

Length of SWContentRegister

`uint8_t version[2]`

Version information of this subcomponent. Version information major.minor, according to java card virtual machine package versions, both values in range 0 to 127. byte 0 is Major value, byte 1 is Minor value.

10.15.1.8 Struct datastore_stream_context_t

- Defined in [File nxp_iot_agent_datastore.h](#)

10.15.1.8.1 Struct Documentation

```
struct datastore_stream_context_t
```

A stream for reading contents from within a datastore, starting at an offset.

Public Members

```
const iot_agent_datastore_t *datastore  
size_t offset
```

10.15.1.9 Struct handle_request_payload_args_t

- Defined in [File nxp_iot_agent_dispatcher.h](#)

10.15.1.9.1 Struct Documentation

```
struct handle_request_payload_args_t
```

Context structure for passing dispatcher information to callbacks for message decoding.

Public Members

```
iot_agent_dispatcher_context_t *dispatcher_context  
iot_agent_response_buffer_t *response_buffer
```

10.15.1.10 Struct iot_agent_datastore_interface_t

- Defined in [File nxp_iot_agent_datastore.h](#)

10.15.1.10.1 Struct Documentation

```
struct iot_agent_datastore_interface_t
```

The interface any datastore needs to implement.

Public Members

```
iot_agent_datastore_allocator_t allocate  
iot_agent_datastore_committer_t commit  
iot_agent_datastore_destroyer_t destroy  
iot_agent_endpoint_interface_t endpoint_interface  
iot_agent_datastore_reader_t read  
iot_agent_datastore_writer_t write
```

10.15.1.11 Struct iot_agent_datastore_t

- Defined in [File nxp_iot_agent_datastore.h](#)

10.15.1.11.1 Struct Documentation

```
struct iot_agent_datastore_t
```

A context holding the state of a datastore; this is passed to datastore interface functions.

Public Members

```
void *context  
uint32_t identifier  
iot_agent_datastore_interface_t iface  
uint32_t type
```

10.15.1.12 Struct iot_agent_dispatcher_context_t

- Defined in [File nxp_iot_agent_dispatcher.h](#)

10.15.1.12.1 Struct Documentation

```
struct iot_agent_dispatcher_context_t
```

Public Members

```
iot_agent_context_t *agent_context  
bool closed  
nxp_iot_EndpointRequest *current_request  
iot_agent_endpoint_cache_table_entry_t endpoints[NXP_IOT_AGENT_MAX_NUM_ENDPOINTS]  
void *network_context  
nxp_iot_ServiceDescriptor *service_descriptor  
nxp_iot_UpdateStatusReport *status_report  
iot_agent_stream_type_t stream_type  
bool successful_crl_verification_done  
bool update_status
```

10.15.1.13 Struct iot_agent_endpoint_cache_table_entry_t

- Defined in [File nxp_iot_agent_dispatcher.h](#)

10.15.1.13.1 Struct Documentation

```
struct iot_agent_endpoint_cache_table_entry_t
```

Public Members

```
void *endpoint_context  
iot_agent_endpoint_interface_t endpoint_interface  
uint32_t id  
nxp_iot_EndpointType type
```

10.15.1.14 Struct iot_agent_keystore_interface_t

- Defined in [File nxp_iot_agent_keystore.h](#)

10.15.1.14.1 Struct Documentation

```
struct iot_agent_keystore_interface_t
```

The interface any keystore needs to implement.

Public Members

```
iot_agent_keystore_session_closer_t close_session  
iot_agent_keystore_destroyer_t destroy  
iot_agent_endpoint_interface_t endpoint_interface  
iot_agent_keystore_session_opener_t open_session
```

10.15.1.15 Struct iot_agent_keystore_key_ref_t

- Defined in [File nxp_iot_agent_keystore.h](#)

10.15.1.15.1 Struct Documentation

```
struct iot_agent_keystore_key_ref_t
```

A structure used to abstract key reference between different OpenSSL versions.

Public Members

```
void *key_ref
```

10.15.1.16 Struct iot_agent_keystore_t

- Defined in [File nxp_iot_agent_keystore.h](#)

10.15.1.16.1 Struct Documentation

```
struct iot_agent_keystore_t
```

A structure binding a keystore interface and a keystore context to a keystore instance.

Public Members

```
void *context  
uint32_t identifier  
iot_agent_keystore_interface_t iface  
uint32_t type
```

10.15.1.17 Struct multicast_package_t

- Defined in [File sems_lite_api.h](#)

10.15.1.17.1 Struct Documentation

struct **multicast_package_t**

Information about package

Public Members

size_t multicastCommandsLen

Length of multicast commands (as pointed by pMulticastCommands). If multicastCommandsLen mismatches to contents of pMulticastCommands, there will be undesired behavior of system.

size_t multicastPackageNameLen

Length of multicast Package name

uint8_t multicastPackageVersion[2]

Version information of this MulticastPackage, describing the sum version over all contained content. In case exactly one SubComponent is contained this version is equal to the version of this SubComponent. Version information major.minor, both values in range 0 to 127. byte 0 is Major value, byte 1 is Minor value.

size_t pkSemsCaAutLen

Length of pPkSemsCaAut

const uint8_t *pMulticastCommands

The complete Multicast Applet Loader Script (certificate, signature, encrypted and signed commands) in protobuf format.

const char *pMulticastPackageName

Giving a descriptive name to the content of the Multicast Package. It is possible that one Multicast Package contains multiple Applet binaries and also that a Multicast Package contains no Applet binary at all (e.g. for deletion).

const uint8_t *pPkSemsCaAut

The PkSemsCaAut key found in the SEMS_SP_AUT.txt file as PK.AP.AUT.

const uint8_t *pSignatureOverCommands

The signature over the multicast commands in an machine readable form. So it does not have to be parsed from the script commands. String encoding is chosen here, as mayna json parsers can not handle such large integer values.

const sub_component_metaData_t *pSubComponentMetaData

A list of subcomponents of this MulticastPackage, designating all Executable Load Files (ELFs) Contained. It usually contains one entry, but can have multiple in the case multiple dependent ELFs get modified. This list can be empty, e.g. for a KeyRotation or content deletion MulticastPackage.

uint32_t requiredFreeBytesNonVolatileMemory

Minimum required free Non Volatile memory in bytes that have to be free on the target device before execution of MulticastPackage.

uint32_t requiredFreeBytesTransientMemory

Minimum required free transient (RAM) memory in bytes that have to be present on the target device before execution of MulticastPackage.

uint32_t semsLiteAPIVersion

Version information of SEMS Lite Agent API Defined follow SEMS_LITE_AGENT_VER_MAJOR

size_t signatureOverCommandsLen

Length of signatureOverCommands

```
uint8_t target12Nc[6]
```

Target 12nc is a 12 digit numerical code identifying the target device where this MulticastPackage is intended to be executed on, as known to customers and used on EdgeLock2Go to identify device types.

```
uint8_t targetEntityID[16]
```

Entity ID, 16bytes long Binary Coded Decimal, of the target device where this MulticastPackage is intended to be executed on. It is an identifier of the key-set of the Multicast Applet Loader. As the key-set and related Entity ID can change over the lifetime of the device it is separated from the Commercial Name. This is stored to have it formatted in the BCD encoded and therefore recognizable form.

10.15.1.18 Struct NXECKey03_StaticCtx_t

- Defined in [File nxScp03_Types.h](#)

10.15.1.18.1 Struct Documentation

```
struct NXECKey03_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

10.15.1.19 Struct NXSCP03_AuthCtx_t

- Defined in [File nxScp03_Types.h](#)

10.15.1.19.1 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

10.15.1.20 Struct NXSCP03_DynCtx_t

- Defined in [File nxScp03_Types.h](#)

10.15.1.20.1 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 different 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

10.15.1.21 Struct NXSCP03_StaticCtx_t

- Defined in [File nxScp03_Types.h](#)

10.15.1.21.1 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

int **key_len**

AES Key size

uint8_t **keyVerNo**

Key version no to use for channel authentication in SCP03

[sss_object_t](#) **Mac**

static secure channel authentication key obj

10.15.1.22 Struct SE05x_Applet_Feature_Disable_t

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.1.22.1 Struct Documentation

```
struct SE05x_Applet_Feature_Disable_t
    Used to disable Applet Features via sss_se05x_set_feature

Public Members
    uint8_t EXTCFG_FORBID_AES_GCM
        Disable feature AES_GCM B8b8
    uint8_t EXTCFG_FORBID_AES_GCM_EXT_IV
        Disable feature AES_GCM_EXT_IV B8b7
    uint8_t EXTCFG_FORBID_ECDH
        Disable feature ECDH B2b8
    uint8_t EXTCFG_FORBID_HKDF_EXTRACT
        Disable feature HKDF_EXTRACT B10b7
    uint8_t EXTCFG_FORBID_RSA_LT_2K
        Disable feature RSA_LT_2K B6b8
    uint8_t EXTCFG_FORBID_RSA_SHA1
        Disable feature RSA_SHA1 B6b7
```

10.15.1.23 Struct SE05x_Applet_Feature_t

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.1.23.1 Struct Documentation

```
struct SE05x_Applet_Feature_t
    Used to enable 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_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

10.15.1.24 Struct Se05x_AppletFeatures_t

- Defined in [File se05x_tlv.h](#)

10.15.1.24.1 Struct Documentation

struct Se05x_AppletFeatures_t

[Se05x_AppletFeatures_t](#) Representing features of SE05x applet

Public Members

[SE05x_ExtendedFeatures_t](#) *extended_features
Pointer to extended_features.

[SE05x_Variant_t](#) variant
Variant of the SE05x applet features.

10.15.1.25 Struct SE05x_AuthCtx_ECKey_t

- Defined in [File nxScp03_Types.h](#)

10.15.1.25.1 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.

10.15.1.26 Struct SE05x_AuthCtx_ID_t

- Defined in [File nxScp03_Types.h](#)

10.15.1.26.1 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

10.15.1.27 Struct SE05x_ExtendedFeatures_t

- Defined in [File se05x_tlv.h](#)

10.15.1.27.1 Struct Documentation

struct **SE05x_ExtendedFeatures_t**

SE05x_ExtendedFeatures_t Representing Extended feature in SE05x

Public Members

uint8_t features[30]

Extended feature array

10.15.1.28 Struct SE05x_I2CM_configData_t

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.1.28.1 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

10.15.1.29 Struct SE05x_I2CM_readData_t

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.1.29.1 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

10.15.1.30 Struct SE05x_I2CM_securityData_t

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.1.30.1 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

10.15.1.31 Struct SE05x_I2CM_structuralIssue_t

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.1.31.1 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

10.15.1.32 Struct SE05x_I2CM_writeData_t

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.1.32.1 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

10.15.1.33 Struct SE05x_TimeStamp_t

- Defined in [File se05x_tlv.h](#)

10.15.1.33.1 Struct Documentation

struct **SE05x_TimeStamp_t**

[**SE05x_TimeStamp_t**](#) Representing timestamp in SE05x

Public Members

 uint8_t **ts[12]**

 TimeStamp Array

10.15.1.34 Struct Se05xApdu_t

- Defined in [File se05x_tlv.h](#)

10.15.1.34.1 Struct Documentation

struct **Se05xApdu_t**

[**Se05xApdu_t**](#) struct representing an APDU for SE05x communication.

Public Members

 uint8_t ***dataToMac**

 Pointer to data for MAC Calculation

 size_t **dataToMacLen**

 Length of data for MAC Calculation

 uint8_t ***se05xCmd**

 SE05x Command

 const [**tlvHeader_t *se05xCmd_hdr**](#)

 SE05x Command Header

 size_t **se05xCmdLC**

 SE05x Command LC

```
size_t se05xCmdLCW
    SE05x Command LC width
size_t se05xCmdLen
    SE05x Command Length
uint8_t *se05xTxBuf
    transmit buffer
size_t se05xTxBufLen
    Length of transmit buffer
size_t ws_LC
    With Session LC
size_t ws_LCW
    With Session LC Width 1 or 3 bytes
uint8_t *wsSe05x_cmd
    WithSession SE05X command
size_t wsSe05x_cmdLen
    WithSession SE05X command Length
uint8_t *wsSe05x_tag1Cmd
    WithSession SE05X Tag1 Command Data
size_t wsSe05x_tag1CmdLen
    WithSession SE05X Tag1 Command Data Len
size_t wsSe05x_tag1Len
    WithSession SE05X Tag1 len
size_t wsSe05x_tag1W
    WithSession SE05X Tag1 Width
```

10.15.1.35 Struct Se05xPolicy_t

- Defined in [File se05x_tlv.h](#)

10.15.1.35.1 Struct Documentation

struct **Se05xPolicy_t**

[Se05xPolicy_t](#) representing policy in SE05x [Se05xPolicy_t](#) structure defines a policy in SE05x with a policy value and its length

Public Members

uint8_t ***value**

policy value

size_t **value_len**

Length of Policy value

10.15.1.36 Struct Se05xSession

- Defined in [File se05x_tlv.h](#)

10.15.1.36.1 Struct Documentation

struct **Se05xSession**

 struct [Se05xSession](#) representing a session in SE05x

Public Members

 uint32_t **applet_version**

 applet version

 uint32_t **auth_id**

 auth ID associated with session

[SE_AuthType_t](#) **authType**

 Type of authentication for the session

 void ***conn_ctx**

 Connection data context

[smStatus_t](#) (*fp_DeDecrypt)(struct [Se05xSession](#) *pSession, size_t prevCmdBufLen, uint8_t *pInRxBuf, size_t *pInRxBufLen, uint8_t hasle)

 API called by fp_TXn. Helps handle Applet/Fast SCP to decrypt buffer.

 But this API never reads any data

[smStatus_t](#) (*fp_RawTXn)(void *conn_ctx, struct [sss_se05x_tunnel_context](#) *pChannelCtx, [SE_AuthType_t](#) currAuth, const [tlvHeader_t](#) *hdr, uint8_t *cmdBuf, size_t cmdBufLen, uint8_t *rsp, size_t *rspLen, uint8_t hasle)

 It's either a minimal/single implementation that calls smCom_TransceiveRaw()

 if pTunnelCtx is Null, directly call smCom_TransceiveRaw()

 Or an API part of tunnel ctx that can do PlatformSCP

[smStatus_t](#) (*fp_Transform)(struct [Se05xSession](#) *pSession, const [tlvHeader_t](#) *inHdr, uint8_t *inCmdBuf, size_t inCmdBufLen, [tlvHeader_t](#) *outHdr, uint8_t *pTxBuf, size_t *pTxBufLen, uint8_t hasle)

 API called by fp_TXn. Helps handle UserID/Applet/ECKey to transform buffer.

 But this API never sends any data out over any communication link.

[smStatus_t](#) (*fp_TXn)(struct [Se05xSession](#) *pSession, const [tlvHeader_t](#) *hdr, uint8_t *cmdBuf, size_t cmdBufLen, uint8_t *rsp, size_t *rspLen, uint8_t hasle)

 Meta Function

 Internal first calls fp_Transform Then calls fp_RawTXn Then calls fp_DeDecrypt

 uint8_t **hasSession**

 Indicating session is active

 uint8_t **logical_channel**

 struct [sss_se05x_tunnel_context](#) ***pChannelCtx**

 pChannelCtx holds the context information for SE05x tunnel communication.

[NXSCP03_DynCtx_t](#) ***pdynScp03Ctx**

 pdynScp03Ctx holds the dynamic context information for SCP03 channel

 uint8_t **value[8]**

 Array of 8 bytes representing session value.

10.15.1.37 Struct SE_Connect_Ctx_t

- Defined in [File nxScp03_Types.h](#)

10.15.1.37.1 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

12C 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

uint8_t sessionResume

Set to 1 if we should resume a session already open with SE

uint16_t sizeOfStucture

to support binary compatibility/check, sizeOfStucture 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

10.15.1.38 Struct sems_lite_agent_ctx_t

- Defined in [File sems_lite_agent_context.h](#)

10.15.1.38.1 Struct Documentation

struct **sems_lite_agent_ctx_t**

This structure seems to manage the context and state of a communication session, possibly involving interactions with multiple applets or components within a system.

Public Members**uint32_t freePHeapCentralGap**

Indicates the amount of free memory in the central gap of the PHeap

uint32_t freeTransient

Represents free transient memory information, potentially related to COD (Card Operating System) information from JCOP.

Se05xSession_t *pS05x_Ctx

Boot context

bool recovery_executed

Flag for get recovery script

bool session_opened

Flag for session status: Open/Close

bool skip_next_commands

Skip following commands

Internal state variable.

When one APDU in this APDU stream has failed, skip through all the next APDUs (do not send them) and return back to the caller.

uint32_t status_word

Status word of last R-APDU.

10.15.1.39 Struct sems_lite_agent_request_ctx_t

- Defined in [File sems_lite_agent_context.h](#)

10.15.1.39.1 Struct Documentation

struct sems_lite_agent_request_ctx_t

This structure likely facilitates managing the state and progress of requests made by the SEMS Lite agent, including keeping track of the current command being processed, checkpoints for managing progress, and retry counts for error handling.

Public Members**int checkpoint_index**

index of the checkpoint, which is a specific point in the sequence of commands that marks progress or a known state.

int current_command_index

index of the current command in the sequence of commands being processed.

int retry_count

number of times a command has been retried in case of failure.

10.15.1.40 Struct sems_lite_SEAppInfoList_t

- Defined in [File sems_lite_api.h](#)

10.15.1.40.1 Struct Documentation

struct **sems_lite_SEAppInfoList_t**

Information of about Applet/Package

See Table 11-36: GlobalPlatform Registry Data (TLV), GPCardSpc_v2.2.pdf

The response from Applet is put to rspBuf

After parsing that response, the pointers to respective members is set and it points to relevant part in rspBuf, this way saving memory. However, the Length is updated so that application use this information.

Public Members

uint8_t **AIDLen**

Length of the Applet ID

uint8_t **LifeCycleState**

Life-cycle state

uint8_t **LoadFileAIDLen**

Length of LoadFileAID.

uint8_t **LoadFileVersionNumberLen**

Length of pLoadFileVersionNumber.

uint8_t ***pAID**

Applet ID

uint8_t ***pLoadFileAID**

Application's Executable Load File AID.

uint8_t ***pLoadFileVersionNumber**

Executable Load File Version Number.

uint8_t ***pPrivileges**

Privileges.

uint8_t **PrivilegesLen**

Length of Privileges

uint8_t ***pSecurityDomainAID**

Associated Security Domain's AID.

uint8_t **rspBuf[(256 + 5)]**

Response from Applet.

size_t **rspBufLen**

Length of response from Applet.

uint8_t **SecurityDomainAIDLen**

Length of SecurityDomainAID.

10.15.1.41 Struct service_link_t

- Defined in [File nxp_iot_agent_service.h](#)

10.15.1.41.1 Struct Documentation

```
struct service_link_t
{
    Public Members
    const iot_agent_datastore_t *datastore
    const iot_agent_keystore_t *keystore
    size_t offset
}
```

10.15.1.42 Struct SM_SECURE_SCP03_KEYOBJ

- Defined in [File nxScp03_Types.h](#)

10.15.1.42.1 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.
}
```

10.15.1.43 Struct sss_aead_t

- Defined in [File fsl_sss_api.h](#)

10.15.1.43.1 Struct Documentation

```
struct sss_aead_t
{
    Authenticated Encryption with Additional Data.
    Public Members
    sss_algorithm_t algorithm
        Algorithm to be used
    uint8_t data[(0 + (5 * sizeof(void*)) + (6 * sizeof(int)) + (5 * sizeof(void*)) + 32)]
    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
        High level operation (encrypt/decrypt)
}
```

sss_session_t ***session**

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

10.15.1.44 Struct sss_asymmetric_t

- Defined in [File fsl_sss_api.h](#)

10.15.1.44.1 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*)) + 32)]

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

10.15.1.45 Struct sss_connect_ctx_t

- Defined in [File nxScp03_Types.h](#)

10.15.1.45.1 Struct Documentation**struct sss_connect_ctx_t**

Wrapper strucutre [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

10.15.1.46 Struct sss_derive_key_t

- Defined in [File fsl_sss_api.h](#)

10.15.1.46.1 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*)) + 32)]

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

10.15.1.47 Struct sss_digest_t

- Defined in [File fsl_sss_api.h](#)

10.15.1.47.1 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*)) + 32)]

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.

10.15.1.48 Struct sss_ecc_point_t

- Defined in [File fsl_sss_api.h](#)

10.15.1.48.1 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

10.15.1.49 Struct sss_eccgfp_group_t

- Defined in [File fsl_sss_api.h](#)

10.15.1.49.1 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

10.15.1.50 Struct sss_key_store_t

- Defined in [File fsl_sss_api.h](#)

10.15.1.50.1 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*)) + 32)]`

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.

10.15.1.51 Struct `sss_mac_t`

- Defined in [File fsl_sss_api.h](#)

10.15.1.51.1 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 (kMode_SSS_Mac)

`sss_session_t *session`

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

10.15.1.52 Struct `sss_object_t`

- Defined in [File fsl_sss_api.h](#)

10.15.1.52.1 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*)) + 32)]
    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 referenced from sss\_key\_part\_t
```

10.15.1.53 Struct sss_policy_asym_key_u

- Defined in [File fsl_sss_policy.h](#)

10.15.1.53.1 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
```

10.15.1.54 Struct sss_policy_common_pcr_value_u

- Defined in [File fsl_sss_policy.h](#)

10.15.1.54.1 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
```

10.15.1.55 Struct sss_policy_common_u

- Defined in [File fsl_sss_policy.h](#)

10.15.1.55.1 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 can_Read
        Allow to read the object
    uint8_t can_Write
        Allow to write the object
    uint8_t forbid_All
        Forbid all operations
    uint8_t req_pcr_val
        Require PCR value
    uint8_t req_Sm
        Require having secure messaging enabled with encryption and integrity on the command
```

10.15.1.56 Struct sss_policy_counter_u

- Defined in [File fsl_sss_policy.h](#)

10.15.1.56.1 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

10.15.1.57 Struct sss_policy_desfire_changekey_authId_value_u

- Defined in [File fsl_sss_policy.h](#)

10.15.1.57.1 Struct Documentation

struct **sss_policy_desfire_changekey_authId_value_u**

DESFire ChangeKey - authentication key identifier.

Public Members

uint32_t **desfire_authId**

DESFire authentication object ID

10.15.1.58 Struct sss_policy_file_u

- Defined in [File fsl_sss_policy.h](#)

10.15.1.58.1 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

10.15.1.59 Struct sss_policy_internal_sign_tbs_value_u

- Defined in [File fsl_sss_policy.h](#)

10.15.1.59.1 Struct Documentation

struct **sss_policy_internal_sign_tbs_value_u**

Allow internal sign.

Public Members

uint32_t **tbsItemList_KeyId**

identifier of the tbsItemList Secure Object

10.15.1.60 Struct **sss_policy_key_drv_master_keyid_value_u**

- Defined in [File fsl_sss_policy.h](#)

10.15.1.60.1 Struct Documentation

struct **sss_policy_key_drv_master_keyid_value_u**

Key Derive - Master key identifier.

Public Members

uint32_t **masterKeyId**

Master key ID

10.15.1.61 Struct **sss_policy_pcr_u**

- Defined in [File fsl_sss_policy.h](#)

10.15.1.61.1 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

10.15.1.62 Struct **sss_policy_session_u**

- Defined in [File fsl_sss_policy.h](#)

10.15.1.62.1 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

10.15.1.63 Struct sss_policy_sym_key_u

- Defined in [File fsl_sss_policy.h](#)

10.15.1.63.1 Struct Documentation

struct sss_policy_sym_key_u

Policies applicable to Symmetric KEY

Public Members**uint8_t allow_kdf_ext_rnd**

Allow kdf(prf) external random

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_Desfire_KD

Allow Desfire key derivation

uint8_t can_Encrypt

Allow encryption

uint8_t can_Gen

Allow to (re)generate the object

uint8_t can_HKDF

Allow HKDF

uint8_t can_Import_Export

Allow to imported or exported

uint8_t can_KA

Allow key agreement. Only for SE051H, key agreement is applicable for symm objects

```
uint8_t can_KD
    Allow key derivation
uint8_t can_PBKDF
    Allow PBKDF
uint8_t can_Sign
    Allow signature generation
uint8_t can_TLS_KDF
    Allow TLS PRF key derivation
uint8_t can_TLS_PMS_KD
    Allow TLS PMS key derivation
uint8_t can_usage_hmac_pepper
    Allow usage as hmac pepper
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
uint8_t forbid_external_iv
    Forbid External iv
```

10.15.1.64 Struct sss_policy_t

- Defined in [File fsl_sss_policy.h](#)

10.15.1.64.1 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

10.15.1.65 Struct sss_policy_u

- Defined in [File fsl_sss_policy.h](#)

10.15.1.65.1 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_desfire_changekey_authId_value_u](#) **desfire_auth_id**

[sss_policy_file_u](#) **file**

[sss_policy_key_drv_master_keyid_value_u](#) **master_key_id**

[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_internal_sign_tbs_value_u](#) **tbsItemList**

[sss_policy_type_u](#) **type**

Secure Object Type

10.15.1.66 Struct sss_policy_userid_u

- Defined in [File fsl_sss_policy.h](#)

10.15.1.66.1 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

10.15.1.67 Struct sss_rng_context_t

- Defined in [File fsl_sss_api.h](#)

10.15.1.67.1 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*)) + 32)]

sss_session_t ***session**

Pointer to the session

10.15.1.68 Struct **sss_se05x_aead_t**

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.1.68.1 Struct Documentation

struct **sss_se05x_aead_t**

Authenticated Encryption with Additional Data.

Public Members

sss_algorithm_t **algorithm**

Algorithm to be used

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**

High level operation (encrypt/decrypt)

sss_se05x_session_t ***session**

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

10.15.1.69 Struct **sss_se05x_asymmetric_t**

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.1.69.1 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

10.15.1.70 Struct sss_se05x_attst_comp_data_t

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.1.70.1 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 **cmd**[100]

capdu for attestation

size_t **cmdLen**

capdu Length of attestation

uint8_t **objSize**[2]

object size

size_t **objSizeLen**

object size Length

uint8_t **signature**[512]

Signature for attestation

size_t **signatureLen**

Length of signature

SE05x_TimeStamp_t **timeStamp**

time stamp
size_t **timeStampLen**
Length of timeStamp

10.15.1.71 Struct sss_se05x_attst_data_t

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.1.71.1 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

10.15.1.72 Struct sss_se05x_derive_key_t

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.1.72.1 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

10.15.1.73 Struct sss_se05x_digest_t

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.1.73.1 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.

10.15.1.74 Struct sss_se05x_key_store_t

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.1.74.1 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

10.15.1.75 Struct sss_se05x_mac_t

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.1.75.1 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
```

10.15.1.76 Struct sss_se05x_rng_context_t

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.1.76.1 Struct Documentation

struct **sss_se05x_rng_context_t**

Random number generator context

Public Members

sss_se05x_session_t *session

Pointer to the session

10.15.1.77 Struct sss_se05x_symmetric_t

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.1.77.1 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 can 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 its 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.

10.15.1.78 Struct sss_session_t

- Defined in [File fsl_sss_api.h](#)

10.15.1.78.1 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*)) + (7 * sizeof(void*)) + 32)]

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

10.15.1.79 Struct **sss_symmetric_t**

- Defined in [File fsl_sss_api.h](#)

10.15.1.79.1 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 + 32)]

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.

10.15.1.80 Struct **sss_tunnel_t**

- Defined in [File fsl_sss_api.h](#)

10.15.1.80.1 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*)) + 32)]

struct sss_tunnel_t::[anonymous] **extension**

Reserved memory for implementation specific extension

sss_session_t ***session**

Pointer to the session

uint32_t **tunnelType**

Tunnel to which Applet (Currently unused)

10.15.1.81 Struct tlvHeader_t

- Defined in [File fsl_sss_api.h](#)

10.15.1.81.1 Struct Documentation

struct **tlvHeader_t**

Header for a IS716 APDU

Public Members

uint8_t **hdr**[0 + 1 + 1 + 1 + 1]

ISO 7816 APDU Header

10.15.2 Enums

10.15.2.1 Enum _iot_agent_stream_type_t

- Defined in [File nxp_iot_agent_dispatcher.h](#)

10.15.2.1.1 Enum Documentation

enum **_iot_agent_stream_type_t**

Values:

enumerator **STREAM_TYPE_NETWORK**

Network stream

enumerator **STREAM_TYPE_BUFFER_REQUESTS**

Buffer stream in Requests format

10.15.2.2 Enum _sems_lite_recovery_status_t

- Defined in [File sems_lite_api.h](#)

10.15.2.2.1 Enum Documentation

enum **_sems_lite_recovery_status_t**

Indicates the current state of recovery process in a System

Values:

enumerator **sems_lite_recovery_not_started**

Recovery Not Started

enumerator **sems_lite_recovery_started**

Recovery Started

10.15.2.3 Enum _sems_lite_status

- Defined in [File sems_lite_agent_common.h](#)

10.15.2.3.1 Enum Documentation

enum **_sems_lite_status**

Status of SEMS Lite update operation

Values:

enumerator **kStatus_SEMS_Lite_Success**

Operation was successful

enumerator **kStatus_SEMS_Lite_ERR_COM**

Communication Error

enumerator **kStatus_SEMS_Lite_ERR_DoReRun**

Update not completed please provide update package again.

enumerator **kStatus_SEMS_Lite_ERR_NotApplicable**

Update not applicable on this Chip/type.

enumerator **kStatus_SEMS_Lite_ERR_DoRecovery**

Update can not be completed. Please provide recovery package, to roll back to last working version.

enumerator **kStatus_SEMS_Lite_ERR_Fatal**

Unresolvable error.

enumerator **kStatus_SEMS_Lite_ERR_NonEnoughNVMemory**

Not enough NV memory.

enumerator **kStatus_SEMS_Lite_ERR_NonEnoughTransientMemory**

Not enough transient memory.

enumerator **kStatus_SEMS_Lite_ERR_MinPreviousVersion**

Current SE version not meet min previous version request.

enumerator **kStatus_SEMS_Lite_ERR_OlderVersion**

Older than current SE version

enumerator **kStatus_SEMS_Lite_ERR_General**

General error.

10.15.2.4 Enum _sems_lite_tearDown_status_t

- Defined in [File sems_lite_api.h](#)

10.15.2.4.1 Enum Documentation

enum _sems_lite_tearDown_status_t

Check Tear down Amd-I section 4.14

Values:

enumerator **sems_lite_notear**

The script has been completely executed

enumerator **sems_lite_tear**

Script execution was interrupted because of teardown.

enumerator **sems_lite_tearDown_status_invalid**

10.15.2.5 Enum _sems_lite_upgradeProgress_status_t

- Defined in [File sems_lite_api.h](#)

10.15.2.5.1 Enum Documentation

enum _sems_lite_upgradeProgress_status_t

Track the progress of an upgrade process in a software system, distinguishing between when the upgrade session is active or not, and providing a special value for invalid states.

Values:

enumerator **sems_lite_upgrade_not_inProgress**

Upgrade session Not in Progress

enumerator **sems_lite_upgrade_inProgress**

Upgrade session In Progress

enumerator **sems_lite_upgrade_invalid**

10.15.2.6 Enum _sems_lite_version_check_result_t

- Defined in [File sems_lite_agent_common.h](#)

10.15.2.6.1 Enum Documentation

enum _sems_lite_version_check_result_t

Status of SEMS Lite version check result

Values:

enumerator **kStatus_SEMS_Lite_Version_Pass**

Operation was successful

enumerator **kStatus_SEMS_Lite_Version_ERR_MIN**

Failed due to min previous version.

enumerator **kStatus_SEMS_Lite_Version_ERR_Downgrade**

Failed due to downgrade.

10.15.2.7 Enum eSE05xType_t

- Defined in [File se05x_tlv.h](#)

10.15.2.7.1 Enum Documentation

enum **eSE05xType_t**

Values:

enumerator **CRED_DEFAULT**
enumerator **CRED_EC**
enumerator **CRED_RSA**
enumerator **CRED_AES**
enumerator **CRED_DES**
enumerator **CRED_BINARY**
enumerator **CRED_PIN**
enumerator **CRED_COUNTER**
enumerator **CRED_PCR**
enumerator **CRED_OBJECT**
enumerator **CRED_PUB_EC**
enumerator **CRED_PUB_RSA**

10.15.2.8 Enum SE05x_AeadAlgo_t

- Defined in [File se05x_enums.h](#)

10.15.2.8.1 Enum Documentation

enum **SE05x_AeadAlgo_t**

AEAD Algorithms

Values:

enumerator **kSE05x_AeadAlgo_NA**
Invalid
enumerator **kSE05x_AeadGCMAlgo**
enumerator **kSE05x_AeadGCM_IVAlgo**
enumerator **kSE05x_AeadCCMAlgo**

10.15.2.9 Enum SE05x_AppletConfig_t

- Defined in [File se05x_enums.h](#)

10.15.2.9.1 Enum Documentation

enum **SE05x_AppletConfig_t**

Features which are available / enabled in the Applet

Values:

enumerator **kSE05x_AppletConfig_NA**

Invalid

enumerator **kSE05x_AppletConfig_ECDSA_ECDH_ECDHE**

EC DSA and DH support

enumerator **kSE05x_AppletConfig_EDDSA**

Use of curve RESERVED_ID_ECC_ED_25519

enumerator **kSE05x_AppletConfig_DH_MONT**

Use of curve RESERVED_ID_ECC_MONT_DH_25519

enumerator **kSE05x_AppletConfig_HMAC**

Writing HMACKey objects

enumerator **kSE05x_AppletConfig_RSA_PLAIN**

Writing RSAKey objects

enumerator **kSE05x_AppletConfig_RSA_CRT**

Writing RSAKey objects

enumerator **kSE05x_AppletConfig_AES**

Writing AESKey objects

enumerator **kSE05x_AppletConfig_DES**

Writing DESKey objects

enumerator **kSE05x_AppletConfig_PBKDF**

PBKDF2

enumerator **kSE05x_AppletConfig_TLS**

TLS Handshake support commands (see 4.16) in APDU Spec

enumerator **kSE05x_AppletConfig_MIFARE**

Mifare DESFire support (see 4.15) in APDU Spec

enumerator **kSE05x_AppletConfig_RFU1**

RFU1

enumerator **kSE05x_AppletConfig_I2CM**

I2C Master support (see 4.17) in APDU Spec

enumerator **kSE05x_AppletConfig_RFU2**

RFU2

10.15.2.10 Enum **SE05x_AppletResID_t**

- Defined in [File se05x_enums.h](#)

10.15.2.10.1 Enum Documentation

enum **SE05x_AppletResID_t**

Reserved identifiers of the Applet

Values:

enumerator **kSE05x_AppletResID_NA**

Invalid

enumerator **kSE05x_AppletResID_TRANSPORT**

An authentication object which allows the user to switch LockState of the applet. The LockState defines whether the applet is transport locked or not.

enumerator **kSE05x_AppletResID_KP_ECKEY_USER**

A device unique NIST P-256 key pair which contains SK.SE.ECKA and PK.SE.ECKA in ECKey session context.

enumerator **kSE05x_AppletResID_KP_ECKEY_IMPORT**

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.

enumerator **kSE05x_AppletResID_FEATURE**

An authentication object which allows the user to change the applet variant.

enumerator **kSE05x_AppletResID_FACTORY_RESET**

An authentication object which allows the user to delete all objects, except trust provisioned by NXP objects.

enumerator **kSE05x_AppletResID_UNIQUE_ID**

A BinaryFile Secure Object which holds the device unique ID. This file cannot be overwritten or deleted.

enumerator **kSE05x_AppletResID_PLATFORM_SCP**

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.

enumerator **kSE05x_AppletResID_I2CM_ACCESS**

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.

enumerator **kSE05x_AppletResID_RESTRICT**

An authentication object which grants access to the SetLockState command

enumerator **kSE05x_AppletResID_SPAKE2P_M_P256_UNCOMPRESSED**

SPAKE2P_M_P256_UNCOMPRESSED KEY

enumerator **kSE05x_AppletResID_SPAKE2P_N_P256_UNCOMPRESSED**

SPAKE2P_N_P256_UNCOMPRESSED KEY

enumerator **kSE05x_AppletResID_SPAKE2P_M_P384_UNCOMPRESSED**

SPAKE2P_M_P384_UNCOMPRESSED KEY

enumerator **kSE05x_AppletResID_SPAKE2P_N_P384_UNCOMPRESSED**

SPAKE2P_N_P384_UNCOMPRESSED KEY

enumerator **kSE05x_AppletResID_SPAKE2P_M_P521_UNCOMPRESSED**

SPAKE2P_M_P521_UNCOMPRESSED KEY

enumerator **kSE05x_AppletResID_SPAKE2P_N_P521_UNCOMPRESSED**

SPAKE2P_N_P521_UNCOMPRESSED KEY

10.15.2.11 Enum SE05x_AttestationAlgo_t

- Defined in [File se05x_enums.h](#)

10.15.2.11.1 Enum Documentation

enum **SE05x_AttestationAlgo_t**

Attestation

Values:

enumerator kSE05x_AttestationAlgo_NA
enumerator kSE05x_AttestationAlgo_EC_PLAIN
enumerator kSE05x_AttestationAlgo_EC_SHA
enumerator kSE05x_AttestationAlgo_EC_SHA_224
enumerator kSE05x_AttestationAlgo_EC_SHA_256
enumerator kSE05x_AttestationAlgo_EC_SHA_384
enumerator kSE05x_AttestationAlgo_EC_SHA_512
enumerator kSE05x_AttestationAlgo_ED25519PURE_SHA_512
enumerator kSE05x_AttestationAlgo_RSA_SHA1_PKCS1_PSS
enumerator kSE05x_AttestationAlgo_RSA_SHA224_PKCS1_PSS
enumerator kSE05x_AttestationAlgo_RSA_SHA256_PKCS1_PSS
enumerator kSE05x_AttestationAlgo_RSA_SHA384_PKCS1_PSS
enumerator kSE05x_AttestationAlgo_RSA_SHA512_PKCS1_PSS
enumerator kSE05x_AttestationAlgo_RSA_SHA_224_PKCS1
enumerator kSE05x_AttestationAlgo_RSA_SHA_256_PKCS1
enumerator kSE05x_AttestationAlgo_RSA_SHA_384_PKCS1
enumerator kSE05x_AttestationAlgo_RSA_SHA_512_PKCS1

10.15.2.12 Enum SE05x_AttestationType_t

- Defined in [File se05x_enums.h](#)

10.15.2.12.1 Enum Documentation

enum **SE05x_AttestationType_t**

In case the read is attested

Values:

enumerator kSE05x_AttestationType_None
enumerator kSE05x_AttestationType_AUTH

10.15.2.13 Enum SE05x_Cipher_Oper_OneShot_t

- Defined in [File se05x_enums.h](#)

10.15.2.13.1 Enum Documentation

enum **SE05x_Cipher_Oper_OneShot_t**

One Shot operations helper

Values:

enumerator **kSE05x_Cipher_Oper_OneShot_NA**

enumerator **kSE05x_Cipher_Oper_OneShot_Encrypt**

enumerator **kSE05x_Cipher_Oper_OneShot_Decrypt**

10.15.2.14 Enum SE05x_Cipher_Oper_t

- Defined in [File se05x_enums.h](#)

10.15.2.14.1 Enum Documentation

enum **SE05x_Cipher_Oper_t**

Cipher Operation.

Encrypt or decrypt

Values:

enumerator **kSE05x_Cipher_Oper_NA**

enumerator **kSE05x_Cipher_Oper_Encrypt**

enumerator **kSE05x_Cipher_Oper_Decrypt**

10.15.2.15 Enum SE05x_CipherMode_t

- Defined in [File se05x_enums.h](#)

10.15.2.15.1 Enum Documentation

enum **SE05x_CipherMode_t**

Symmetric cipher modes

Values:

enumerator **kSE05x_CipherMode_NA**

Invalid

enumerator **kSE05x_CipherMode_DES_CBC_NOPAD**

Typically using DESKey identifiers

enumerator **kSE05x_CipherMode_DES_CBC_ISO9797_M1**

Typically using DESKey identifiers

enumerator **kSE05x_CipherMode_DES_CBC_ISO9797_M2**

Typically using DESKey identifiers

enumerator **kSE05x_CipherMode_DES_CBC_PKCS5**
NOT SUPPORTED

enumerator **kSE05x_CipherMode_DES_ECB_NOPAD**
Typically using DESKey identifiers

enumerator **kSE05x_CipherMode_DES_ECB_ISO9797_M1**
NOT SUPPORTED

enumerator **kSE05x_CipherMode_DES_ECB_ISO9797_M2**
NOT SUPPORTED

enumerator **kSE05x_CipherMode_DES_ECB_PKCS5**
NOT SUPPORTED

enumerator **kSE05x_CipherMode_AES_ECB_NOPAD**
Typically using AESKey identifiers

enumerator **kSE05x_CipherMode_AES_CBC_NOPAD**
Typically using AESKey identifiers

enumerator **kSE05x_CipherMode_AES_CBC_ISO9797_M1**
Typically using AESKey identifiers

enumerator **kSE05x_CipherMode_AES_CBC_ISO9797_M2**
Typically using AESKey identifiers

enumerator **kSE05x_CipherMode_AES_CBC_PKCS5**
NOT SUPPORTED

enumerator **kSE05x_CipherMode_AES_GCM**
Typically using AEAD GCM mode

enumerator **kSE05x_CipherMode_AES_CTR**
Typically using AESKey identifiers

enumerator **kSE05x_CipherMode_AES_CTR_INT_IV**
Typically using AESKey CTR mode with internal IV Gen Only used by MW. Change to kSE05x_CipherMode_AES_CTR when sending to SE

enumerator **kSE05x_CipherMode_AES_GCM_INT_IV**
Typically using AEAD GCM with internal IV Gen

enumerator **kSE05x_CipherMode_AES_CCM**
Typically using AEAD CCM mode

enumerator **kSE05x_CipherMode_AES_CCM_INT_IV**
Typically using AEAD CCM with internal IV Gen

10.15.2.16 Enum SE05x_CryptoContext_t

- Defined in [File se05x_enums.h](#)

10.15.2.16.1 Enum Documentation

enum **SE05x_CryptoContext_t**

Cryptographic context for operation

Values:

enumerator **kSE05x_CryptoContext_NA**

 Invalid

enumerator **kSE05x_CryptoContext_DIGEST**

 For DigestInit/DigestUpdate/DigestFinal

enumerator **kSE05x_CryptoContext_CIPHER**

 For CipherInit/CipherUpdate/CipherFinal

enumerator **kSE05x_CryptoContext_SIGNATURE**

 For MACInit/MACUpdate/MACFinal

enumerator **kSE05x_CryptoContext_AEAD**

 For AEADInit/AEADUpdate/AEADFinal

enumerator **kSE05x_CryptoContext_PAKE**

 For PAKE

10.15.2.17 Enum **SE05x_CryptoObject_t**

- Defined in [File se05x_enums.h](#)

10.15.2.17.1 Enum Documentation

enum **SE05x_CryptoObject_t**

Crypto object identifiers

Values:

enumerator **kSE05x_CryptoObject_NA**

 Invalid

enumerator **kSE05x_CryptoObject_DIGEST_SHA**

enumerator **kSE05x_CryptoObject_DIGEST_SHA224**

enumerator **kSE05x_CryptoObject_DIGEST_SHA256**

enumerator **kSE05x_CryptoObject_DIGEST_SHA384**

enumerator **kSE05x_CryptoObject_DIGEST_SHA512**

enumerator **kSE05x_CryptoObject_DES_CBC_NOPAD**

enumerator **kSE05x_CryptoObject_DES_CBC_ISO9797_M1**

enumerator **kSE05x_CryptoObject_DES_CBC_ISO9797_M2**

enumerator **kSE05x_CryptoObject_DES_CBC_PKCS5**

enumerator **kSE05x_CryptoObject_DES_ECB_NOPAD**

enumerator **kSE05x_CryptoObject_DES_ECB_ISO9797_M1**

enumerator **kSE05x_CryptoObject_DES_ECB_ISO9797_M2**

enumerator **kSE05x_CryptoObject_DES_ECB_PKCS5**

```
enumerator kSE05x_CryptoObject_AES_ECB_NOPAD
enumerator kSE05x_CryptoObject_AES_CBC_NOPAD
enumerator kSE05x_CryptoObject_AES_CBC_ISO9797_M1
enumerator kSE05x_CryptoObject_AES_CBC_ISO9797_M2
enumerator kSE05x_CryptoObject_AES_CBC_PKCS5
enumerator kSE05x_CryptoObject_AES_CTR
enumerator kSE05x_CryptoObject_AES_CTR_INT_IV
enumerator kSE05x_CryptoObject_HMAC_SHA1
enumerator kSE05x_CryptoObject_HMAC_SHA256
enumerator kSE05x_CryptoObject_HMAC_SHA384
enumerator kSE05x_CryptoObject_HMAC_SHA512
enumerator kSE05x_CryptoObject_CMAC_128
enumerator kSE05x_CryptoObject_AES_GCM
enumerator kSE05x_CryptoObject_AES_GCM_INT_IV
enumerator kSE05x_CryptoObject_AES_CCM
enumerator kSE05x_CryptoObject_AES_CCM_INT_IV
enumerator kSE05x_CryptoObject_PAKE_TYPE_A
enumerator kSE05x_CryptoObject_PAKE_TYPE_B
enumerator kSE05x_CryptoObject_End
```

10.15.2.18 Enum SE05x_DigestMode_t

- Defined in [File se05x_enums.h](#)

10.15.2.18.1 Enum Documentation

```
enum SE05x_DigestMode_t
    Hashing/Digest algorithms
    Values:
        enumerator kSE05x_DigestMode_NA
            Invalid
        enumerator kSE05x_DigestMode_NO_HASH
        enumerator kSE05x_DigestMode_SHA
        enumerator kSE05x_DigestMode_SHA224
            Not supported
        enumerator kSE05x_DigestMode_SHA256
        enumerator kSE05x_DigestMode_SHA384
        enumerator kSE05x_DigestMode_SHA512
```

10.15.2.19 Enum SE05x_ECCurve_t

- Defined in [File se05x_enums.h](#)

10.15.2.19.1 Enum Documentation

enum **SE05x_ECCurve_t**

ECC Curve Identifiers

Values:

enumerator **kSE05x_ECCurve_NA**

 Invalid

enumerator **kSE05x_ECCurve_NIST_P192**

enumerator **kSE05x_ECCurve_NIST_P224**

enumerator **kSE05x_ECCurve_NIST_P256**

enumerator **kSE05x_ECCurve_NIST_P384**

enumerator **kSE05x_ECCurve_NIST_P521**

enumerator **kSE05x_ECCurve_Brainpool160**

enumerator **kSE05x_ECCurve_Brainpool192**

enumerator **kSE05x_ECCurve_Brainpool224**

enumerator **kSE05x_ECCurve_Brainpool256**

enumerator **kSE05x_ECCurve_Brainpool320**

enumerator **kSE05x_ECCurve_Brainpool384**

enumerator **kSE05x_ECCurve_Brainpool512**

enumerator **kSE05x_ECCurve_Secp160k1**

enumerator **kSE05x_ECCurve_Secp192k1**

enumerator **kSE05x_ECCurve_Secp224k1**

enumerator **kSE05x_ECCurve_Secp256k1**

enumerator **kSE05x_ECCurve TPM_ECC_BN_P256**

enumerator **kSE05x_ECCurve_ECC_ED_25519**

 Not Weierstrass

enumerator **kSE05x_ECCurve_ECC_MONT_DH_25519**

enumerator **kSE05x_ECCurve_ECC_MONT_DH_448**

 Not Weierstrass

10.15.2.20 Enum SE05x_ECCurveParam_t

- Defined in [File se05x_enums.h](#)

10.15.2.20.1 Enum Documentation

enum **SE05x_ECCurveParam_t**

Parameters while setting the curve

Values:

enumerator **kSE05x_ECCurveParam_NA**

 Invalid

enumerator **kSE05x_ECCurveParam_PARAM_A**

enumerator **kSE05x_ECCurveParam_PARAM_B**

enumerator **kSE05x_ECCurveParam_PARAM_G**

enumerator **kSE05x_ECCurveParam_PARAM_N**

enumerator **kSE05x_ECCurveParam_PARAM_PRIME**

10.15.2.21 Enum **SE05x_ECDHAlgo_t**

- Defined in [File se05x_enums.h](#)

10.15.2.21.1 Enum Documentation

enum **SE05x_ECDHAlgo_t**

Different ECDH algorithms

Values:

enumerator **kSE05x_ECDHAlgo_NA**

 Invalid

enumerator **kSE05x_ECDHAlgo_EC_SVDP_DH**

 Generates the SHA1 of the X coordinate.

enumerator **kSE05x_ECDHAlgo_EC_SVDP_DH_PLAIN**

 Generates the X coordinate.

10.15.2.22 Enum **SE05x_ECPMAlgo_t**

- Defined in [File se05x_enums.h](#)

10.15.2.22.1 Enum Documentation

enum **SE05x_ECPMAlgo_t**

ECPMAlgo

Values:

enumerator **kSE05x_ECPMAlgo_PACE_GM**

enumerator **kSE05x_ECPMAlgo_SVDP_DH_PLAIN_XY**

10.15.2.23 Enum **SE05x_ECSignatureAlgo_t**

- Defined in [File se05x_enums.h](#)

10.15.2.23.1 Enum Documentation

enum **SE05x_ECSignatureAlgo_t**

Different signature algorithms for EC

Values:

enumerator **kSE05x_ECSignatureAlgo_NA**

 Invalid

enumerator **kSE05x_ECSignatureAlgo_PLAIN**

 NOT SUPPORTED

enumerator **kSE05x_ECSignatureAlgo_SHA**

enumerator **kSE05x_ECSignatureAlgo_SHA_224**

enumerator **kSE05x_ECSignatureAlgo_SHA_256**

enumerator **kSE05x_ECSignatureAlgo_SHA_384**

enumerator **kSE05x_ECSignatureAlgo_SHA_512**

10.15.2.24 Enum **SE05x_EDSignatureAlgo_t**

- Defined in [File se05x_enums.h](#)

10.15.2.24.1 Enum Documentation

enum **SE05x_EDSignatureAlgo_t**

Different signature algorithms for ED

Values:

enumerator **kSE05x_EDSignatureAlgo_NA**

 Invalid

enumerator **kSE05x_EDSignatureAlgo_ED25519PURE_SHA_512**

 Message input must be plain Data. Pure EDDSA algorithm

10.15.2.25 Enum **SE05x_HealthCheckMode_t**

- Defined in [File se05x_enums.h](#)

10.15.2.25.1 Enum Documentation

enum **SE05x_HealthCheckMode_t**

Health check

Values:

enumerator **kSE05x_HealthCheckMode_NA**

 Invalid

enumerator **kSE05x_HealthCheckMode_FIPS**

 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.

enumerator kSE05x_HealthCheckMode_CODE_SIGNATURE

Performs ROM integrity checks. When the test fails, the chip triggers the attack counter and the chip will reset.

enumerator kSE05x_HealthCheckMode_DYNAMIC_FLASH_INTEGRITY

Performs flash integrity tests. When the test fails, the chip triggers the attack counter and the chip will reset.

enumerator kSE05x_HealthCheckMode_SHIELDING

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.

enumerator kSE05x_HealthCheckMode_SENSOR

Performs self-tests on hardware sensors and reports the status.

enumerator kSE05x_HealthCheckMode_SFR_CHECK

Performs self-tests on the hardware registers. When the test fails, the chip triggers the attack counter and the chip will reset.

10.15.2.26 Enum SE05x_HkdfMode_t

- Defined in [File se05x_enums.h](#)

10.15.2.26.1 Enum Documentation

enum SE05x_HkdfMode_t

HKDF Mode

Values:

enumerator kSE05x_HkdfMode_NA

Invalid

enumerator kSE05x_HkdfMode_ExtractExpand**enumerator kSE05x_HkdfMode_ExpandOnly**

10.15.2.27 Enum SE05x_I2CM_Baud_Rate_t

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.2.27.1 Enum Documentation

enum SE05x_I2CM_Baud_Rate_t

Configuration for I2CM

Values:

enumerator kSE05x_I2CM_Baud_Rate_100Khz**enumerator kSE05x_I2CM_Baud_Rate_400Khz**

10.15.2.28 Enum SE05x_I2CM_securityReq_t

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.2.28.1 Enum Documentation

enum **SE05x_I2CM_securityReq_t**

Additional operation on data read by I2C

Values:

enumerator **kSE05x_Security_None**

enumerator **kSE05x_Sign_Request**

enumerator **kSE05x_Sign_Enc_Request**

10.15.2.29 Enum SE05x_I2CM_status_t

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.2.29.1 Enum Documentation

enum **SE05x_I2CM_status_t**

Status of I2CM Transaction

Values:

enumerator **kSE05x_I2CM_Success**

enumerator **kSE05x_I2CM_I2C_Nack_Fail**

enumerator **kSE05x_I2CM_I2C_Write_Error**

enumerator **kSE05x_I2CM_I2C_Read_Error**

enumerator **kSE05x_I2CM_I2C_Time_Out_Error**

enumerator **kSE05x_I2CM_Invalid_Tag**

enumerator **kSE05x_I2CM_Invalid_Length**

enumerator **kSE05x_I2CM_Invalid_Length_Encode**

enumerator **kSE05x_I2CM_I2C_Config**

10.15.2.30 Enum SE05x_I2CM_TAG_t

- Defined in [File se05x_enums.h](#)

10.15.2.30.1 Enum Documentation

enum **SE05x_I2CM_TAG_t**

I2C Master micro operation.

Values:

enumerator **kSE05x_TAG_I2CM_Config**

enumerator **kSE05x_TAG_I2CM_Write**

enumerator **kSE05x_TAG_I2CM_Read**

10.15.2.31 Enum SE05x_I2CM_TLV_type_t

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.2.31.1 Enum Documentation

enum **SE05x_I2CM_TLV_type_t**

Types of entries in an I2CM Transaction

Values:

enumerator **kSE05x_I2CM_None**

Do nothing

enumerator **kSE05x_I2CM_Configure**

Configure the address, baudrate

enumerator **kSE05x_I2CM_Write**

Write to I2C Slave

enumerator **kSE05x_I2CM_Read**

Read from I2C Slave

enumerator **kSE05x_I2CM_StructuralIssue**

Response from SE05x that there is something wrong

10.15.2.32 Enum SE05x_INS_t

- Defined in [File se05x_enums.h](#)

10.15.2.32.1 Enum Documentation

enum **SE05x_INS_t**

Values for INS in ISO7816 APDU

Values:

enumerator **kSE05x_INS_NA**

Invalid

enumerator **kSE05x_INS_MASK_INS_CHAR**

3 MSBit for instruction characteristics.

enumerator **kSE05x_INS_MASK_INSTRUCTION**

5 LSBit for instruction

enumerator **kSE05x_INS_TRANSIENT**

Mask for transient object creation, can only be combined with INS_WRITE.

enumerator **kSE05x_INS_AUTH_OBJECT**

Mask for authentication object creation, can only be combined with INS_WRITE

enumerator **kSE05x_INS_ATTEST**

Mask for getting attestation data.

enumerator **kSE05x_INS_WRITE**

Write or create a persistent object.

enumerator **kSE05x_INS_READ**

Read the object

enumerator **kSE05x_INS_CRYPTO**

Perform Security Operation

enumerator **kSE05x_INS_MGMT**

General operation

enumerator **kSE05x_INS_PROCESS**

Process session command

10.15.2.33 Enum SE05x_KeyPart_t

- Defined in [File se05x_enums.h](#)

10.15.2.33.1 Enum Documentation

enum **SE05x_KeyPart_t**

Part of the asymmetric key

Values:

enumerator **kSE05x_KeyPart_NA**

enumerator **kSE05x_KeyPart_Pair**

Key pair (private key + public key)

enumerator **kSE05x_KeyPart_Private**

Private key

enumerator **kSE05x_KeyPart_Public**

Public key

10.15.2.34 Enum SE05x_LockIndicator_t

- Defined in [File se05x_enums.h](#)

10.15.2.34.1 Enum Documentation

enum **SE05x_LockIndicator_t**

Transient / Persistent lock

Values:

enumerator **kSE05x_LockIndicator_NA**

Invalid

enumerator **kSE05x_LockIndicator_TRANSIENT_LOCK**

enumerator **kSE05x_LockIndicator_PERSISTENT_LOCK**

10.15.2.35 Enum SE05x_LockState_t

- Defined in [File se05x_enums.h](#)

10.15.2.35.1 Enum Documentation

enum **SE05x_LockState_t**

Lock the sample (until unlocked)

Values:

enumerator **kSE05x_LockState_NA**

 Invalid

enumerator **kSE05x_LockState_LOCKED**

10.15.2.36 Enum SE05x_Mac_Oper_t

- Defined in [File se05xEnums.h](#)

10.15.2.36.1 Enum Documentation

enum **SE05x_Mac_Oper_t**

MAC operations

Values:

enumerator **kSE05x_Mac_Oper_NA**

enumerator **kSE05x_Mac_Oper_Generate**

enumerator **kSE05x_Mac_Oper_Validate**

10.15.2.37 Enum SE05x_MACAlgo_t

- Defined in [File se05xEnums.h](#)

10.15.2.37.1 Enum Documentation

enum **SE05x_MACAlgo_t**

HMAC/CMAC Algorithms

Values:

enumerator **kSE05x_MACAlgo_NA**

 Invalid

enumerator **kSE05x_MACAlgo_HMAC_SHA1**

enumerator **kSE05x_MACAlgo_HMAC_SHA256**

enumerator **kSE05x_MACAlgo_HMAC_SHA384**

enumerator **kSE05x_MACAlgo_HMAC_SHA512**

enumerator **kSE05x_MACAlgo_CMAC_128**

enumerator **kSE05x_MACAlgo_DES_CMAC8**

10.15.2.38 Enum SE05x_MemoryType_t

- Defined in [File se05xEnums.h](#)

10.15.2.38.1 Enum Documentation

enum **SE05x_MemoryType_t**

Data for available memory

Values:

enumerator **kSE05x_MemoryType_NA**

Invalid

enumerator **kSE05x_MemoryType_PERSISTENT**

Persistent memory

enumerator **kSE05x_MemoryType_TRANSIENT_RESET**

Transient memory, clear on reset

enumerator **kSE05x_MemoryType_TRANSIENT_DESELECT**

Transient memory, clear on deselect

10.15.2.39 Enum **SE05x_MoreIndicator_t**

- Defined in [File se05x_enums.h](#)

10.15.2.39.1 Enum Documentation

enum **SE05x_MoreIndicator_t**

When there are more entries yet to be fetched from few of the APIs

Values:

enumerator **kSE05x_MoreIndicator_NA**

Invalid

enumerator **kSE05x_MoreIndicator_NO_MORE**

No more data available

enumerator **kSE05x_MoreIndicator_MORE**

More data available

10.15.2.40 Enum **SE05x-Origin_t**

- Defined in [File se05x_enums.h](#)

10.15.2.40.1 Enum Documentation

enum **SE05x-Origin_t**

Where was this object originated

Values:

enumerator **kSE05x-Origin_NA**

Invalid

enumerator **kSE05x-Origin_EXTERNAL**

Generated outside the module.

enumerator **kSE05x-Origin_INTERNAL**

Generated inside the module.

enumerator **kSE05x_Origin_PROVISIONED**

Trust provisioned by NXP

10.15.2.41 Enum SE05x_P1_t

- Defined in [File se05x_enums.h](#)

10.15.2.41.1 Enum Documentation

enum **SE05x_P1_t**

Values for P1 in ISO7816 APDU

Values:

enumerator **kSE05x_P1_NA**

Invalid

enumerator **kSE05x_P1_UNUSED**

Highest bit not used

enumerator **kSE05x_P1_MASK_KEY_TYPE**

2 MSBit for key type

enumerator **kSE05x_P1_MASK_CRED_TYPE**

5 LSBit for credential type

enumerator **kSE05x_P1_KEY_PAIR**

Key pair (private key + public key)

enumerator **kSE05x_P1_PRIVATE**

Private key

enumerator **kSE05x_P1_PUBLIC**

Public key

enumerator **kSE05x_P1_DEFAULT**

enumerator **kSE05x_P1_EC**

enumerator **kSE05x_P1_RSA**

enumerator **kSE05x_P1_AES**

enumerator **kSE05x_P1_DES**

enumerator **kSE05x_P1_HMAC**

enumerator **kSE05x_P1_BINARY**

enumerator **kSE05x_P1_UserID**

enumerator **kSE05x_P1_COUNTER**

enumerator **kSE05x_P1_PCR**

enumerator **kSE05x_P1_CURVE**

enumerator **kSE05x_P1_SIGNATURE**

```
enumerator kSE05x_P1_MAC
enumerator kSE05x_P1_CIPHER
enumerator kSE05x_P1_TLS
enumerator kSE05x_P1_CRYPTO_OBJ
enumerator kSE05x_P1_AEAD
    Applet >= 4.4
enumerator kSE05x_P1_AEAD_SP800_38D
    Applet >= 4.4
enumerator kSE05x_P1_PAKE
```

10.15.2.42 Enum SE05x_P2_t

- Defined in [File se05x_enums.h](#)

10.15.2.42.1 Enum Documentation

```
enum SE05x_P2_t
    Values for P2 in ISO7816 APDU
    Values:
        enumerator kSE05x_P2_DEFAULT
            Invalid
        enumerator kSE05x_P2_GENERATE
        enumerator kSE05x_P2_CREATE
        enumerator kSE05x_P2_SIZE
        enumerator kSE05x_P2_SIGN
        enumerator kSE05x_P2_VERIFY
        enumerator kSE05x_P2_INIT
        enumerator kSE05x_P2_UPDATE
        enumerator kSE05x_P2_FINAL
        enumerator kSE05x_P2_ONESHOT
        enumerator kSE05x_P2_DH
        enumerator kSE05x_P2_DIVERSIFY
        enumerator kSE05x_P2_AUTH_FIRST_PART2
        enumerator kSE05x_P2_AUTH_NONFIRST_PART2
        enumerator kSE05x_P2_DUMP_KEY
        enumerator kSE05x_P2_CHANGE_KEY_PART1
        enumerator kSE05x_P2_CHANGE_KEY_PART2
        enumerator kSE05x_P2_KILL_AUTH
```

```
enumerator kSE05x_P2_IMPORT
enumerator kSE05x_P2_EXPORT
enumerator kSE05x_P2_SESSION_CREATE
enumerator kSE05x_P2_SESSION_CLOSE
enumerator kSE05x_P2_SESSION_REFRESH
enumerator kSE05x_P2_SESSION_POLICY
enumerator kSE05x_P2_VERSION
enumerator kSE05x_P2_VERSION_EXT
enumerator kSE05x_P2_MEMORY
enumerator kSE05x_P2_LIST
enumerator kSE05x_P2_TYPE
enumerator kSE05x_P2_EXIST
enumerator kSE05x_P2_DELETE_OBJECT
enumerator kSE05x_P2_DELETE_ALL
enumerator kSE05x_P2_SESSION_UserID
enumerator kSE05x_P2_HKDF
enumerator kSE05x_P2_PBKDF
enumerator kSE05x_P2_HKDF_EXPAND_ONLY
enumerator kSE05x_P2_I2CM
enumerator kSE05x_P2_I2CM_ATTESTED
enumerator kSE05x_P2_MAC
enumerator kSE05x_P2_UNLOCK_CHALLENGE
enumerator kSE05x_P2_CURVE_LIST
enumerator kSE05x_P2_ID
enumerator kSE05x_P2_ENCRYPT_ONESHOT
enumerator kSE05x_P2_DECRYPT_ONESHOT
enumerator kSE05x_P2_ATTEST
enumerator kSE05x_P2_ATTRIBUTES
enumerator kSE05x_P2_CPLC
enumerator kSE05x_P2_TIME
enumerator kSE05x_P2_TRANSPORT
enumerator kSE05x_P2_VARIANT
enumerator kSE05x_P2_PARAM
enumerator kSE05x_P2_DELETE_CURVE
enumerator kSE05x_P2_ENCRYPT
```

```
enumerator kSE05x_P2_DECRYPT
enumerator kSE05x_P2_VALIDATE
enumerator kSE05x_P2_GENERATE_ONESHOT
enumerator kSE05x_P2_VALIDATE_ONESHOT
enumerator kSE05x_P2_CRYPTO_LIST
enumerator kSE05x_P2_RANDOM
enumerator kSE05x_P2_TLS_PMS
enumerator kSE05x_P2_TLS_PRF_CLI_HELLO
enumerator kSE05x_P2_TLS_PRF_SRV_HELLO
enumerator kSE05x_P2_TLS_PRF_CLI_RND
enumerator kSE05x_P2_TLS_PRF_SRV_RND
enumerator kSE05x_P2_TLS_PRF_BOTH
enumerator kSE05x_P2_RAW
enumerator kSE05x_P2_IMPORT_EXT
enumerator kSE05x_P2_SCP
enumerator kSE05x_P2_AUTH_FIRST_PART1
enumerator kSE05x_P2_AUTH_NONFIRST_PART1
enumerator kSE05x_P2_CM_COMMAND
enumerator kSE05x_P2_MODE_OF_OPERATION
enumerator kSE05x_P2_RESTRICT
enumerator kSE05x_P2_SANITY
enumerator kSE05x_P2_DH_REVERSE
enumerator kSE05x_P2_READ_STATE
enumerator kSE05x_P2_ECPM
```

10.15.2.43 Enum SE05x_PAKEMode_t

- Defined in [File se05x_enums.h](#)

10.15.2.43.1 Enum Documentation

enum **SE05x_PAKEMode_t**

PAKE Mode

Values:

enumerator kSE05x_SPAKE2PLUS_NA

 Invalid

enumerator kSE05x_SPAKE2PLUS_P256_SHA256_HKDF_HMAC

enumerator kSE05x_SPAKE2PLUS_P256_SHA512_HKDF_HMAC

```
enumerator kSE05x_SPAKE2PLUS_P384_SHA256_HKDF_HMAC
enumerator kSE05x_SPAKE2PLUS_P384_SHA512_HKDF_HMAC
enumerator kSE05x_SPAKE2PLUS_P521_SHA512_HKDF_HMAC
enumerator kSE05x_SPAKE2PLUS_P256_SHA256_HKDF_CMAC
enumerator kSE05x_SPAKE2PLUS_P256_SHA512_HKDF_CMAC
```

10.15.2.44 Enum SE05x_PAKEState_t

- Defined in [File se05x_enums.h](#)

10.15.2.44.1 Enum Documentation

enum SE05x_PAKEState_t

PAKE State

Values:

```
enumerator kSE05x_PAKE_STATE_SETUP
enumerator kSE05x_PAKE_STATE_KEY_SHARE_GENERATED
enumerator kSE05x_PAKE_STATE_SESSION_KEYS_GENERATED
```

10.15.2.45 Enum SE05x_PlatformSCPRequest_t

- Defined in [File se05x_enums.h](#)

10.15.2.45.1 Enum Documentation

enum SE05x_PlatformSCPRequest_t

Mandate platform SCP or not

Values:

```
enumerator kSE05x_PlatformSCPRequest_NA
    Invalid
enumerator kSE05x_PlatformSCPRequest_REQUIRED
    Platform SCP is required (full enc & MAC)
enumerator kSE05x_PlatformSCPRequest_NOT_REQUIRED
    No platform SCP required.
```

10.15.2.46 Enum SE05x_RestrictMode_t

- Defined in [File se05x_enums.h](#)

10.15.2.46.1 Enum Documentation

enum SE05x_RestrictMode_t

Applet >= 4.4

See [Se05x_API_DisableObjCreation](#)

Values:

enumerator **kSE05x_RestrictMode_NA**
enumerator **kSE05x_RestrictMode_RESTRICT_NEW**
enumerator **kSE05x_RestrictMode_RESTRICT_ALL**

10.15.2.47 Enum SE05x_Result_t

- Defined in [File se05x_enums.h](#)

10.15.2.47.1 Enum Documentation

enum **SE05x_Result_t**

Result of operations

Values:

enumerator **kSE05x_Result_NA**
Invalid
enumerator **kSE05x_Result_SUCCESS**
enumerator **kSE05x_Result_FAILURE**

10.15.2.48 Enum SE05x_RSABitLength_t

- Defined in [File se05x_enums.h](#)

10.15.2.48.1 Enum Documentation

enum **SE05x_RSABitLength_t**

Size of RSA Key Objects

Values:

enumerator **kSE05x_RSABitLength_NA**
Invalid
enumerator **kSE05x_RSABitLength_512**
enumerator **kSE05x_RSABitLength_1024**
enumerator **kSE05x_RSABitLength_1152**
enumerator **kSE05x_RSABitLength_2048**
enumerator **kSE05x_RSABitLength_3072**
enumerator **kSE05x_RSABitLength_4096**

10.15.2.49 Enum SE05x_RSAEncryptionAlgo_t

- Defined in [File se05x_enums.h](#)

10.15.2.49.1 Enum Documentation

enum **SE05x_RSAEncryptionAlgo_t**

Different encryption/decryption algorithms for RSA

Values:

enumerator **kSE05x_RSAEncryptionAlgo_NA**

 Invalid

enumerator **kSE05x_RSAEncryptionAlgo_NO_PAD**

 Plain RSA, padding required on host.

enumerator **kSE05x_RSAEncryptionAlgo_PKCS1**

 RFC8017: RSAES-PKCS1-v1_5

enumerator **kSE05x_RSAEncryptionAlgo_PKCS1_OAEP**

 RFC8017: RSAES-OAEP

10.15.2.50 Enum **SE05x_RSAKeyComponent_t**

- Defined in [File se05x_enums.h](#)

10.15.2.50.1 Enum Documentation

enum **SE05x_RSAKeyComponent_t**

Part of the RSA Key Objects

Values:

enumerator **kSE05x_RSAKeyComponent_NA**

 Invalid

enumerator **kSE05x_RSAKeyComponent_MOD**

 Modulus

enumerator **kSE05x_RSAKeyComponent_PUB_EXP**

 Public key exponent

enumerator **kSE05x_RSAKeyComponent_PRIV_EXP**

 Private key exponent

enumerator **kSE05x_RSAKeyComponent_P**

 CRT component p

enumerator **kSE05x_RSAKeyComponent_Q**

 CRT component q

enumerator **kSE05x_RSAKeyComponent_DP**

 CRT component dp

enumerator **kSE05x_RSAKeyComponent_DQ**

 CRT component dq

enumerator **kSE05x_RSAKeyComponent_INVQ**

 CRT component q_inv

10.15.2.51 Enum SE05x_RSAKeyFormat_t

- Defined in [File se05x_enums.h](#)

10.15.2.51.1 Enum Documentation

enum **SE05x_RSAKeyFormat_t**

RSA Key format

Values:

enumerator **kSE05x_RSAKeyFormat_CRT**

enumerator **kSE05x_RSAKeyFormat_RAW**

10.15.2.52 Enum SE05x_RSAPubKeyComp_t

- Defined in [File se05x_enums.h](#)

10.15.2.52.1 Enum Documentation

enum **SE05x_RSAPubKeyComp_t**

Public part of RSA Keys

Values:

enumerator **kSE05x_RSAPubKeyComp_NA**

enumerator **kSE05x_RSAPubKeyComp_MOD**

enumerator **kSE05x_RSAPubKeyComp_PUB_EXP**

10.15.2.53 Enum SE05x_RSASignAlgo_t

- Defined in [File se05x_enums.h](#)

10.15.2.53.1 Enum Documentation

enum **SE05x_RSASignAlgo_t**

Algorithms for RSA Signature

Values:

enumerator **kSE05x_RSASignAlgo_NA**

Invalid

enumerator **kSE05x_RSASignAlgo_SHA1_PKCS1_PSS**

RFC8017: RSASSA-PSS

enumerator **kSE05x_RSASignAlgo_SHA224_PKCS1_PSS**

RFC8017: RSASSA-PSS

enumerator **kSE05x_RSASignAlgo_SHA256_PKCS1_PSS**

RFC8017: RSASSA-PSS

enumerator **kSE05x_RSASignAlgo_SHA384_PKCS1_PSS**

RFC8017: RSASSA-PSS

enumerator **kSE05x_RSASignAlgo_SHA512_PKCS1_PSS**

RFC8017: RSASSA-PSS

enumerator **kSE05x_RSASignAlgo_SHA_224_PKCS1**

RFC8017: RSASSA-PKCS1-v1_5

enumerator **kSE05x_RSASignAlgo_SHA_256_PKCS1**

RFC8017: RSASSA-PKCS1-v1_5

enumerator **kSE05x_RSASignAlgo_SHA_384_PKCS1**

RFC8017: RSASSA-PKCS1-v1_5

enumerator **kSE05x_RSASignAlgo_SHA_512_PKCS1**

RFC8017: RSASSA-PKCS1-v1_5

10.15.2.54 Enum SE05x_RSASignatureAlgo_t

- Defined in [File se05x_enums.h](#)

10.15.2.54.1 Enum Documentation

enum **SE05x_RSASignatureAlgo_t**

Different signature algorithms for RSA

Values:

enumerator **kSE05x_RSASignatureAlgo_NA**

Invalid

enumerator **kSE05x_RSASignatureAlgo_SHA1_PKCS1_PSS**

RFC8017: RSASSA-PSS

enumerator **kSE05x_RSASignatureAlgo_SHA224_PKCS1_PSS**

RFC8017: RSASSA-PSS

enumerator **kSE05x_RSASignatureAlgo_SHA256_PKCS1_PSS**

RFC8017: RSASSA-PSS

enumerator **kSE05x_RSASignatureAlgo_SHA384_PKCS1_PSS**

RFC8017: RSASSA-PSS

enumerator **kSE05x_RSASignatureAlgo_SHA512_PKCS1_PSS**

RFC8017: RSASSA-PSS

enumerator **kSE05x_RSASignatureAlgo_SHA1_PKCS1**

RFC8017: RSASSA-PKCS1-v1_5

enumerator **kSE05x_RSASignatureAlgo_SHA_224_PKCS1**

RFC8017: RSASSA-PKCS1-v1_5

enumerator **kSE05x_RSASignatureAlgo_SHA_256_PKCS1**

RFC8017: RSASSA-PKCS1-v1_5

enumerator **kSE05x_RSASignatureAlgo_SHA_384_PKCS1**

RFC8017: RSASSA-PKCS1-v1_5

enumerator **kSE05x_RSASignatureAlgo_SHA_512_PKCS1**

RFC8017: RSASSA-PKCS1-v1_5

10.15.2.55 Enum SE05x_SecObjTyp_t

- Defined in [File se05x_enums.h](#)

10.15.2.55.1 Enum Documentation

enum SE05x_SecObjTyp_t

Type of Object

Values:

```
enumerator kSE05x_SecObjTyp_NA
enumerator kSE05x_SecObjTyp_EC_KEY_PAIR
enumerator kSE05x_SecObjTyp_EC_PRIV_KEY
enumerator kSE05x_SecObjTyp_EC_PUB_KEY
enumerator kSE05x_SecObjTyp_RSA_KEY_PAIR
enumerator kSE05x_SecObjTyp_RSA_KEY_PAIR_CRT
enumerator kSE05x_SecObjTyp_RSA_PRIV_KEY
enumerator kSE05x_SecObjTyp_RSA_PRIV_KEY_CRT
enumerator kSE05x_SecObjTyp_RSA_PUB_KEY
enumerator kSE05x_SecObjTyp_AES_KEY
enumerator kSE05x_SecObjTyp_DES_KEY
enumerator kSE05x_SecObjTyp_BINARY_FILE
enumerator kSE05x_SecObjTyp_UserID
enumerator kSE05x_SecObjTyp_COUNTER
enumerator kSE05x_SecObjTyp_PCR
enumerator kSE05x_SecObjTyp_CURVE
enumerator kSE05x_SecObjTyp_HMAC_KEY
enumerator kSE05x_SecObjTyp_EC_KEY_PAIR_NIST_P192
enumerator kSE05x_SecObjTyp_EC_PRIV_KEY_NIST_P192
enumerator kSE05x_SecObjTyp_EC_PUB_KEY_NIST_P192
enumerator kSE05x_SecObjTyp_EC_KEY_PAIR_NIST_P224
enumerator kSE05x_SecObjTyp_EC_PRIV_KEY_NIST_P224
enumerator kSE05x_SecObjTyp_EC_PUB_KEY_NIST_P224
enumerator kSE05x_SecObjTyp_EC_KEY_PAIR_NIST_P256
enumerator kSE05x_SecObjTyp_EC_PRIV_KEY_NIST_P256
enumerator kSE05x_SecObjTyp_EC_PUB_KEY_NIST_P256
enumerator kSE05x_SecObjTyp_EC_KEY_PAIR_NIST_P384
```

```
enumerator kSE05x_SecObjTyp_EC_PRIV_KEY_NIST_P384
enumerator kSE05x_SecObjTyp_EC_PUB_KEY_NIST_P384
enumerator kSE05x_SecObjTyp_EC_KEY_PAIR_NIST_P521
enumerator kSE05x_SecObjTyp_EC_PRIV_KEY_NIST_P521
enumerator kSE05x_SecObjTyp_EC_PUB_KEY_NIST_P521
enumerator kSE05x_SecObjTyp_EC_KEY_PAIR_Brainpool160
enumerator kSE05x_SecObjTyp_EC_PRIV_KEY_Brainpool160
enumerator kSE05x_SecObjTyp_EC_PUB_KEY_Brainpool160
enumerator kSE05x_SecObjTyp_EC_KEY_PAIR_Brainpool192
enumerator kSE05x_SecObjTyp_EC_PRIV_KEY_Brainpool192
enumerator kSE05x_SecObjTyp_EC_PUB_KEY_Brainpool192
enumerator kSE05x_SecObjTyp_EC_KEY_PAIR_Brainpool224
enumerator kSE05x_SecObjTyp_EC_PRIV_KEY_Brainpool224
enumerator kSE05x_SecObjTyp_EC_PUB_KEY_Brainpool224
enumerator kSE05x_SecObjTyp_EC_KEY_PAIR_Brainpool256
enumerator kSE05x_SecObjTyp_EC_PRIV_KEY_Brainpool256
enumerator kSE05x_SecObjTyp_EC_PUB_KEY_Brainpool256
enumerator kSE05x_SecObjTyp_EC_KEY_PAIR_Brainpool320
enumerator kSE05x_SecObjTyp_EC_PRIV_KEY_Brainpool320
enumerator kSE05x_SecObjTyp_EC_PUB_KEY_Brainpool320
enumerator kSE05x_SecObjTyp_EC_KEY_PAIR_Brainpool384
enumerator kSE05x_SecObjTyp_EC_PRIV_KEY_Brainpool384
enumerator kSE05x_SecObjTyp_EC_PUB_KEY_Brainpool384
enumerator kSE05x_SecObjTyp_EC_KEY_PAIR_Brainpool512
enumerator kSE05x_SecObjTyp_EC_PRIV_KEY_Brainpool512
enumerator kSE05x_SecObjTyp_EC_PUB_KEY_Brainpool512
enumerator kSE05x_SecObjTyp_EC_KEY_PAIR_Secp160k1
enumerator kSE05x_SecObjTyp_EC_PRIV_KEY_Secp160k1
enumerator kSE05x_SecObjTyp_EC_PUB_KEY_Secp160k1
enumerator kSE05x_SecObjTyp_EC_KEY_PAIR_Secp192k1
enumerator kSE05x_SecObjTyp_EC_PRIV_KEY_Secp192k1
enumerator kSE05x_SecObjTyp_EC_PUB_KEY_Secp192k1
enumerator kSE05x_SecObjTyp_EC_KEY_PAIR_Secp224k1
enumerator kSE05x_SecObjTyp_EC_PRIV_KEY_Secp224k1
enumerator kSE05x_SecObjTyp_EC_PUB_KEY_Secp224k1
```

```
enumerator kSE05x_SecObjTyp_EC_KEY_PAIR_Secp256k1
enumerator kSE05x_SecObjTyp_EC_PRIV_KEY_Secp256k1
enumerator kSE05x_SecObjTyp_EC_PUB_KEY_Secp256k1
enumerator kSE05x_SecObjTyp_EC_KEY_PAIR_BN_P256
enumerator kSE05x_SecObjTyp_EC_PRIV_KEY_BN_P256
enumerator kSE05x_SecObjTyp_EC_PUB_KEY_BN_P256
enumerator kSE05x_SecObjTyp_EC_KEY_PAIR_ED25519
enumerator kSE05x_SecObjTyp_EC_PRIV_KEY_ED25519
enumerator kSE05x_SecObjTyp_EC_PUB_KEY_ED25519
enumerator kSE05x_SecObjTyp_EC_KEY_PAIR_MONT_DH_25519
enumerator kSE05x_SecObjTyp_EC_PRIV_KEY_MONT_DH_25519
enumerator kSE05x_SecObjTyp_EC_PUB_KEY_MONT_DH_25519
enumerator kSE05x_SecObjTyp_EC_KEY_PAIR_MONT_DH_448
enumerator kSE05x_SecObjTyp_EC_PRIV_KEY_MONT_DH_448
enumerator kSE05x_SecObjTyp_EC_PUB_KEY_MONT_DH_448
```

10.15.2.56 Enum SE05x_SetIndicator_t

- Defined in [File se05x_enums.h](#)

10.15.2.56.1 Enum Documentation

enum SE05x_SetIndicator_t

Whether object attribute is set

Values:

```
enumerator kSE05x_SetIndicator_NA
    Invalid
enumerator kSE05x_SetIndicator_NOT_SET
enumerator kSE05x_SetIndicator_SET
```

10.15.2.57 Enum SE05x_SPAKE2PlusDeviceType_t

- Defined in [File se05x_enums.h](#)

10.15.2.57.1 Enum Documentation

enum SE05x_SPAKE2PlusDeviceType_t

SPAKE device type

Values:

```
enumerator kSE05x_SPAKE2PLUS_DEVICE_TYPE_UNKNOWN
    Invalid
```

enumerator **SE05x_SPAKE2PLUS_DEVICE_TYPE_A**

Spake device commionsioner

enumerator **SE05x_SPAKE2PLUS_DEVICE_TYPE_B**

Spake device Node/accessory

10.15.2.58 Enum **SE05x_SW12_t**

- Defined in [File se05x_enums.h](#)

10.15.2.58.1 Enum Documentation

enum **SE05x_SW12_t**

Mapping of 2 byte return code

Values:

enumerator **kSE05x_SW12_NA**

Invalid

enumerator **kSE05x_SW12_NO_ERROR**

No Error

enumerator **kSE05x_SW12_CONDITIONS_NOT_SATISFIED**

Conditions not satisfied

enumerator **kSE05x_SW12_SECURITY_STATUS**

Security status not satisfied.

enumerator **kSE05x_SW12_WRONG_DATA**

Wrong data provided.

enumerator **kSE05x_SW12_DATA_INVALID**

Data invalid - policy set invalid for the given object

enumerator **kSE05x_SW12_COMMAND_NOT_ALLOWED**

Command not allowed - access denied based on object policy

10.15.2.59 Enum **SE05x_SymmKeyType_t**

- Defined in [File se05x_enums.h](#)

10.15.2.59.1 Enum Documentation

enum **SE05x_SymmKeyType_t**

Symmetric keys

Values:

enumerator **kSE05x_SymmKeyType_NA**

enumerator **kSE05x_SymmKeyType_AES**

enumerator **kSE05x_SymmKeyType_DES**

enumerator **kSE05x_SymmKeyType_HMAC**

enumerator **kSE05x_SymmKeyType_CMAC**

10.15.2.60 Enum **SE05x_T4T_Access_Ctrl_t**

- Defined in [File se05x_enums.h](#)

10.15.2.60.1 Enum Documentation

enum **SE05x_T4T_Access_Ctrl_t**

T4T Access Control constants

Values:

enumerator **kSE05x_T4T_AccessCtrl_Granted**

Access_Control_Granted

enumerator **kSE05x_T4T_AccessCtrl_Denied**

Access_Control_Denied

enumerator **kSE05x_T4T_AccessCtrl_Locked**

Access_Control_Locked

10.15.2.61 Enum **SE05x_T4T_INS_t**

- Defined in [File se05x_enums.h](#)

10.15.2.61.1 Enum Documentation

enum **SE05x_T4T_INS_t**

Values for INS in ISO7816 APDU

Values:

enumerator **kSE05x_T4T_INS_NA**

Invalid

enumerator **kSE05x_T4T_INS_SELECT**

INS Select

enumerator **kSE05x_T4T_INS_READ_BINARY**

INS Read Binary

enumerator **kSE05x_T4T_INS_UPDATE_BINARY**

INS Update Binary

enumerator **kSE05x_T4T_GET_DATA**

INS Get Data

enumerator **kSE05x_T4T_INS_CONFIGURE_ACCESS_CTRL**

INS Configure access control

enumerator **kSE05x_T4T_INS_MANAGE_READ_CTR**

INS Manage read counter

10.15.2.62 Enum SE05x_T4T_Interface_Const_t

- Defined in [File se05x_enums.h](#)

10.15.2.62.1 Enum Documentation

enum **SE05x_T4T_Interface_Const_t**

T4T Interface constants

Values:

enumerator **kSE05x_T4T_Interface_Contact**

Interface contact

enumerator **kSE05x_T4T_Interface_Contactless**

Interface contactless

10.15.2.63 Enum SE05x_T4T_Operation_Const_t

- Defined in [File se05x_enums.h](#)

10.15.2.63.1 Enum Documentation

enum **SE05x_T4T_Operation_Const_t**

T4T Operation constants

Values:

enumerator **kSE05x_T4T_Operation_Read**

Operation read

enumerator **kSE05x_T4T_Operation_Write**

Operation Write

10.15.2.64 Enum SE05x_T4T_Read_Ctr_Operation_t

- Defined in [File se05x_enums.h](#)

10.15.2.64.1 Enum Documentation

enum **SE05x_T4T_Read_Ctr_Operation_t**

T4T Read counter operation

Values:

enumerator **kSE05x_T4T_Read_Counter_Reset**

Read Counter Reset

enumerator **kSE05x_T4T_Read_Counter_Enable**

Read Counter Enable

enumerator **kSE05x_T4T_Read_Counter_Disable**

Read Counter Disable

10.15.2.65 Enum SE05x_TAG_t

- Defined in [File se05x_enums.h](#)

10.15.2.65.1 Enum Documentation

enum **SE05x_TAG_t**

Different TAG Values to talk to SE05X IoT Applet

Values:

enumerator **kSE05x_TAG_NA**

 Invalid

enumerator **kSE05x_TAG_SESSION_ID**

enumerator **kSE05x_TAG_POLICY**

enumerator **kSE05x_TAG_MAX_ATTEMPTS**

enumerator **kSE05x_TAG_IMPORT_AUTH_DATA**

enumerator **kSE05x_TAG_IMPORT_AUTH_KEY_ID**

enumerator **kSE05x_TAG_POLICY_CHECK**

enumerator **kSE05x_TAG_1**

enumerator **kSE05x_TAG_2**

enumerator **kSE05x_TAG_3**

enumerator **kSE05x_TAG_4**

enumerator **kSE05x_TAG_5**

enumerator **kSE05x_TAG_6**

enumerator **kSE05x_TAG_7**

enumerator **kSE05x_TAG_8**

enumerator **kSE05x_TAG_9**

enumerator **kSE05x_TAG_10**

enumerator **kSE05x_TAG_11**

enumerator **kSE05x_TAG_TIMESTAMP**

enumerator **kSE05x_TAG_SIGNATURE**

enumerator **kSE05x_GP_TAG_CONTRL_REF_PARM**

enumerator **kSE05x_GP_TAG_AID**

enumerator **kSE05x_GP_TAG_KEY_TYPE**

enumerator **kSE05x_GP_TAG_KEY_LEN**

enumerator **kSE05x_GP_TAG_GET_DATA**

enumerator **kSE05x_GP_TAG_DR_SE**

enumerator **kSE05x_GP_TAG_RECEIPT**

enumerator **kSE05x_GP_TAG SCP_PARMS**

10.15.2.66 Enum **SE05x_TLSPerformPRFType_t**

- Defined in [File se05x_enums.h](#)

10.15.2.66.1 Enum Documentation

enum **SE05x_TLSPerformPRFType_t**

TLS Perform PRF

Values:

enumerator **kSE05x_TLS_PRF_NA**

enumerator **kSE05x_TLS_PRF_CLI_HELLO**

enumerator **kSE05x_TLS_PRF_SRV_HELLO**

enumerator **kSE05x_TLS_PRF_CLI_RND**

enumerator **kSE05x_TLS_PRF_SRV_RND**

enumerator **kSE05x_TLS_PRF_BOTH**

10.15.2.67 Enum **SE05x_TransientIndicator_t**

- Defined in [File se05x_enums.h](#)

10.15.2.67.1 Enum Documentation

enum **SE05x_TransientIndicator_t**

Whether object is transient or persistent

Values:

enumerator **kSE05x_TransientIndicator_NA**

Invalid

enumerator **kSE05x_TransientIndicator_PERSISTENT**

enumerator **kSE05x_TransientIndicator_TRANSIENT**

10.15.2.68 Enum **SE05x_TransientType_t**

- Defined in [File se05x_enums.h](#)

10.15.2.68.1 Enum Documentation

enum **SE05x_TransientType_t**

Whether key is transient or persistent

Values:

enumerator **kSE05x_TransientType_Persistent**

enumerator **kSE05x_TransientType_Transient**

10.15.2.69 Enum SE_AuthType_t

- Defined in [File nxScp03_Types.h](#)

10.15.2.69.1 Enum Documentation

enum **SE_AuthType_t**

Defining authentication types in SE05x

Values:

enumerator **kSSS_AuthType_None**

No authentication

enumerator **kSSS_AuthType SCP03**

Global platform SCP03

enumerator **kSSS_AuthType_ID**

(e.g. SE05X) UserID based connection

enumerator **kSSS_AuthType_AESKey**

(e.g. SE05X) Use AESKey for user authentication

Earlier this was called kSSS_AuthType_AppletSCP03

enumerator **kSSS_AuthType_ECKey**

(e.g. SE05X) Use ECKey for user authentication

Earlier this was called kSSS_AuthType_FastSCP

enumerator **kSSS_AuthType_INT_ECKey_Counter**

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.

enumerator **kSSS_SIZE**

10.15.2.70 Enum smStatus_t

- Defined in [File se05x_tlv.h](#)

10.15.2.70.1 Enum Documentation

enum **smStatus_t**

Values:

enumerator **SM_NOT_OK**

enumerator **SM_OK**

enumerator **SM_ERR_WRONG_LENGTH**

enumerator **SM_ERR_CONDITIONS_NOT_SATISFIED**

enumerator **SM_ERR_COMMAND_NOT_ALLOWED**

enumerator **SM_ERR_SECURITY_STATUS**

enumerator **SM_ERR_WRONG_DATA**

enumerator **SM_ERR_DATA_INVALID**

enumerator **SM_ERR_FILE_FULL**
enumerator **SM_ERR_APDU_THROUGHPUT**

10.15.2.71 Enum `sss_access_permission_t`

- Defined in [File fsl_sss_api.h](#)

10.15.2.71.1 Enum Documentation

enum `sss_access_permission_t`

Permissions of an object

Values:

enumerator **kAccessPermission_SSS_Read**

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.

enumerator **kAccessPermission_SSS_Write**

Can change the value of an object

enumerator **kAccessPermission_SSS_Use**

Can use an object

enumerator **kAccessPermission_SSS_Delete**

Can delete an object

enumerator **kAccessPermission_SSS_ChangeAttributes**

Can change permissions applicable to an object

enumerator **kAccessPermission_SSS_All_Permission**

Bitwise OR of all `sss_access_permission`.

10.15.2.72 Enum `sss_algorithm_t`

- Defined in [File fsl_sss_api.h](#)

10.15.2.72.1 Enum Documentation

enum `sss_algorithm_t`

Cryptographic algorithm to be applied

Values:

enumerator **kAlgorithm_None**

enumerator **kAlgorithm_SSS_AES_ECB**

enumerator **kAlgorithm_SSS_AES_CBC**

enumerator **kAlgorithm_SSS_AES_CTR**

enumerator **kAlgorithm_SSS_AES_GCM**

enumerator **kAlgorithm_SSS_AES_CCM**

```
enumerator kAlgorithm_SSS_AES_GCM_INT_IV
enumerator kAlgorithm_SSS_AES_CTR_INT_IV
enumerator kAlgorithm_SSS_AES_CCM_INT_IV
enumerator kAlgorithm_SSS_CHACHA_POLY
enumerator kAlgorithm_SSS_DES_ECB
enumerator kAlgorithm_SSS_DES_CBC
enumerator kAlgorithm_SSS_DES_CBC_ISO9797_M1
enumerator kAlgorithm_SSS_DES_CBC_ISO9797_M2
enumerator kAlgorithm_SSS_DES3_ECB
enumerator kAlgorithm_SSS_DES3_CBC
enumerator kAlgorithm_SSS_DES3_CBC_ISO9797_M1
enumerator kAlgorithm_SSS_DES3_CBC_ISO9797_M2
enumerator kAlgorithm_SSS_SHA1
enumerator kAlgorithm_SSS_SHA224
enumerator kAlgorithm_SSS_SHA256
enumerator kAlgorithm_SSS_SHA384
enumerator kAlgorithm_SSS_SHA512
enumerator kAlgorithm_SSS_CMAC_AES
enumerator kAlgorithm_SSS_HMAC_SHA1
enumerator kAlgorithm_SSS_HMAC_SHA224
enumerator kAlgorithm_SSS_HMAC_SHA256
enumerator kAlgorithm_SSS_HMAC_SHA384
enumerator kAlgorithm_SSS_HMAC_SHA512
enumerator kAlgorithm_SSS_DES_CMAC8
enumerator kAlgorithm_SSS_DH
enumerator kAlgorithm_SSS_ECDH
enumerator kAlgorithm_SSS_DSA_SHA1
enumerator kAlgorithm_SSS_DSA_SHA224
enumerator kAlgorithm_SSS_DSA_SHA256
enumerator kAlgorithm_SSS_RSASSA_PKCS1_V1_5_NO_HASH
enumerator kAlgorithm_SSS_RSASSA_PKCS1_V1_5_SHA1
enumerator kAlgorithm_SSS_RSASSA_PKCS1_V1_5_SHA224
enumerator kAlgorithm_SSS_RSASSA_PKCS1_V1_5_SHA256
enumerator kAlgorithm_SSS_RSASSA_PKCS1_V1_5_SHA384
enumerator kAlgorithm_SSS_RSASSA_PKCS1_V1_5_SHA512
```

```
enumerator kAlgorithm_SSS_RSASSA_PKCS1_PSS_MGF1_SHA1
enumerator kAlgorithm_SSS_RSASSA_PKCS1_PSS_MGF1_SHA224
enumerator kAlgorithm_SSS_RSASSA_PKCS1_PSS_MGF1_SHA256
enumerator kAlgorithm_SSS_RSASSA_PKCS1_PSS_MGF1_SHA384
enumerator kAlgorithm_SSS_RSASSA_PKCS1_PSS_MGF1_SHA512
enumerator kAlgorithm_SSS_RSAES_PKCS1_OAEP_SHA1
enumerator kAlgorithm_SSS_RSAES_PKCS1_OAEP_SHA224
enumerator kAlgorithm_SSS_RSAES_PKCS1_OAEP_SHA256
enumerator kAlgorithm_SSS_RSAES_PKCS1_OAEP_SHA384
enumerator kAlgorithm_SSS_RSAES_PKCS1_OAEP_SHA512
enumerator kAlgorithm_SSS_RSAES_PKCS1_V1_5
enumerator kAlgorithm_SSS_RSASSA_NO_PADDING
enumerator kAlgorithm_SSS_ECDSA_SHA1
enumerator kAlgorithm_SSS_ECDSA_SHA224
enumerator kAlgorithm_SSS_ECDSA_SHA256
enumerator kAlgorithm_SSS_ECDSA_SHA384
enumerator kAlgorithm_SSS_ECDSA_SHA512
```

10.15.2.73 Enum sss_cipher_type_t

- Defined in [File fsl_sss_api.h](#)

10.15.2.73.1 Enum Documentation

enum sss_cipher_type_t

For all cipher types, key bit length is provided at the time key is inserted/generated
Values:

```
enumerator kSSS_CipherType_NONE
enumerator kSSS_CipherType_AES
enumerator kSSS_CipherType_DES
enumerator kSSS_CipherType_CMAC
enumerator kSSS_CipherType_HMAC
enumerator kSSS_CipherType_MAC
enumerator kSSS_CipherType_RSA
enumerator kSSS_CipherType_RSA_CRT
```

RSA RAW format

```
enumerator kSSS_CipherType_EC_NIST_P
```

RSA CRT format

enumerator **kSSS_CipherType_EC_NIST_K**
 Keys Part of NIST-P Family
enumerator **kSSS_CipherType_EC_MONTGOMERY**
 Keys Part of NIST-K Family Montgomery Key,
enumerator **kSSS_CipherType_EC_TWISTED_ED**
 twisted Edwards form elliptic curve public key
enumerator **kSSS_CipherType_EC_BRAINPOOL**
 Brainpool form elliptic curve public key
enumerator **kSSS_CipherType_UserID**
enumerator **kSSS_CipherType_Certificate**
 Use kSSS_CipherType_Binary to store Certificate
enumerator **kSSS_CipherType_Binary**
enumerator **kSSS_CipherType_Count**
enumerator **kSSS_CipherType_PCR**
enumerator **kSSS_CipherType_ReservedPin**

10.15.2.74 Enum **sss_connection_type_t**

- Defined in [File fsl_sss_api.h](#)

10.15.2.74.1 Enum Documentation

enum **sss_connection_type_t**
 Destintion connection type
 Values:
 enumerator **kSSS_ConnectionType_Plain**
 enumerator **kSSS_ConnectionType_Password**
 enumerator **kSSS_ConnectionType_Encrypted**

10.15.2.75 Enum **sss_key_object_mode_t**

- Defined in [File fsl_sss_api.h](#)

10.15.2.75.1 Enum Documentation

enum **sss_key_object_mode_t**
 Persistent / Non persistent mode of a key
 Values:
 enumerator **kKeyObject_Mode_None**
 kKeyObject_Mode_None
 enumerator **kKeyObject_Mode_Persistent**
 Key object will be persisted in memory and will retain it's value after a closed session

enumerator kKeyObject_Mode_Transient

Key Object will be stored in RAM. It will lose its contents after a session is closed

10.15.2.76 Enum sss_key_part_t

- Defined in [File fsl_sss_api.h](#)

10.15.2.76.1 Enum Documentation**enum sss_key_part_t**

Part of a key

Values:

enumerator kSSS_KeyPart_NONE**enumerator kSSS_KeyPart_Default**

Applicable where we have UserID, Binary Files, Certificates, Symmetric Keys, PCR, HMAC-key, counter

enumerator kSSS_KeyPart_Public

Public part of asymmetric key

enumerator kSSS_KeyPart_Private

Private only part of asymmetric key

enumerator kSSS_KeyPart_Pair

Both, public and private part of asymmetric key

10.15.2.77 Enum sss_key_store_prop_a8_t

- Defined in [File fsl_sss_api.h](#)

10.15.2.77.1 Enum Documentation**enum sss_key_store_prop_a8_t**

properties of a Key Store that return array

Values:

enumerator kSSS_KeyStoreProp_a8_Optional_Start

Optional Properties Start

10.15.2.78 Enum sss_mode_t

- Defined in [File fsl_sss_api.h](#)

10.15.2.78.1 Enum Documentation**enum sss_mode_t**

High level algorithmic operations.

Augmented by [sss_algorithm_t](#)

Values:

enumerator kMode_SSS_Encrypt

Encrypt.

enumerator **kMode_SSS_Decrypt**

Decrypt.

enumerator **kMode_SSS_Sign**

Sign.

enumerator **kMode_SSS_Verify**

Verify.

enumerator **kMode_SSS_ComputeSharedSecret**

enumerator **kMode_SSS_Digest**

Message Digest.

enumerator **kMode_SSS_Mac**

Message Authentication Code.

enumerator **kMode_SSS_HKDF_ExpandOnly**

HKDF Expand Only (RFC 5869)

enumerator **kMode_SSS_HKDF_ExtractExpand**

HKDF Extract and Expand (RFC 5869)

enumerator **kMode_SSS_Mac_Validate**

MAC Validate.

10.15.2.79 Enum sss_policy_type_u

- Defined in [File fsl_sss_policy.h](#)

10.15.2.79.1 Enum Documentation

enum **sss_policy_type_u**

Type of policy

Values:

enumerator **KPolicy_None**

No policy applied

enumerator **KPolicy_Session**

Policy related to session.

See Also:

[sss_policy_session_u](#)

enumerator **KPolicy_Sym_Key**

Policy related to key.

See Also:

[sss_policy_key_u](#)

enumerator **KPolicy_Asym_Key**

enumerator **KPolicy_UserID**

enumerator **KPolicy_File**

```
enumerator KPolicy_Counter  
enumerator KPolicy_PCR  
enumerator KPolicy_Common  
enumerator KPolicy_Common_PCR_Value  
enumerator KPolicy_Desfire_Changekey_Auth_Id  
enumerator KPolicy_Derive_Master_Key_Id  
enumerator KPolicy_Internal_Sign
```

10.15.2.80 Enum sss_s05x_sesion_prop_au8_t

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.2.80.1 Enum Documentation

enum sss_s05x_sesion_prop_au8_t

SE050 Properties that can be represented as an array

Values:

```
enumerator kSSS_SE05x_SessionProp_CertUID
```

10.15.2.81 Enum sss_s05x_sesion_prop_u32_t

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.2.81.1 Enum Documentation

enum sss_s05x_sesion_prop_u32_t

SE050 Properties that can be represented as 32bit numbers

Values:

```
enumerator kSSS_SE05x_SessionProp_CertUIDLen
```

10.15.2.82 Enum sss_session_prop_au8_t

- Defined in [File fsl_sss_api.h](#)

10.15.2.82.1 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:

```
enumerator kSSS_SessionProp_au8_NA
```

Invalid

```
enumerator kSSS_SessionProp_szName
```

Name of the product, string

enumerator **kSSS_SessionProp_UID**

Unique Identifier

enumerator **kSSS_SessionProp_au8_Optional_Start**

Optional Properties Start

enumerator **kSSS_SessionProp_au8_Proprietary_Start**

Proprietary Properties Start

10.15.2.83 Enum `sss_session_prop_u32_t`

- Defined in [File fsl_sss_api.h](#)

10.15.2.83.1 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:

enumerator **kSSS_SessionProp_u32_NA**

Invalid

enumerator **kSSS_SessionProp_VerMaj**

Major version

enumerator **kSSS_SessionProp_VerMin**

Minor Version

enumerator **kSSS_SessionProp_VerDev**

Development Version

enumerator **kSSS_SessionProp_UIDLen**

enumerator **kSSS_SessionProp_u32_Optional_Start**

Optional Properties Start

enumerator **kSSS_KeyStoreProp_FreeMem_Persistant**

How much persistent memory is free

enumerator **kSSS_KeyStoreProp_FreeMem_Transient**

How much transient memory is free

enumerator **kSSS_SessionProp_u32_Proprietary_Start**

Proprietary Properties Start

10.15.2.84 Enum `sss_status_t`

- Defined in [File fsl_sss_api.h](#)

10.15.2.84.1 Enum Documentation

enum **sss_status_t**

Status of the SSS APIs

Values:

enumerator **kStatus_SSS_Success**

Operation was successful

enumerator **kStatus_SSS_Fail**

Operation failed

enumerator **kStatus_SSS_InvalidArgument**

Operation not performed because some of the passed parameters were found inappropriate

enumerator **kStatus_SSS_ResourceBusy**

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.

enumerator **kStatus_SSS_ApduThroughputError**

APDU Throughput error

10.15.2.85 Enum **sss_tunnel_dest_t**

- Defined in [File fsl_sss_api.h](#)

10.15.2.85.1 Enum Documentation

enum **sss_tunnel_dest_t**

Entity on the other side of the tunnel

Values:

enumerator **kSSS_TunnelDest_None**

Default value

enumerator **kSSS_TunnelType_Se05x_Iot_applet**

SE05X IoT Applet

10.15.2.86 Enum **sss_type_t**

- Defined in [File fsl_sss_api.h](#)

10.15.2.86.1 Enum Documentation

enum **sss_type_t**

Cryptographic sub system

Values:

enumerator **kType_SSS_SubSystem_NONE**

enumerator **kType_SSS_Software**

Software based

enumerator **kType_SSS_mbedTLS**

```
enumerator kType_SSS_OpenSSL
enumerator kType_SSS_HW
    HOST HW Based
enumerator kType_SSS_SECO
enumerator kType_SSS_Isolated_HW
    Isolated HW
enumerator kType_SSS_Sentinel
enumerator kType_SSS_Sentinel200
enumerator kType_SSS_Sentinel300
enumerator kType_SSS_Sentinel400
enumerator kType_SSS_Sentinel500
enumerator kType_SSS_SecureElement
    Secure Element
enumerator kType_SSS_SE_A71CH
    To connect to https://www.nxp.com/products/:A71CH
enumerator kType_SSS_SE_A71CL
enumerator kType_SSS_SE_SE05x
    To connect to https://www.nxp.com/products/:SE050
enumerator kType_SSS_SubSystem_LAST
```

10.15.3 Unions

10.15.3.1 Union SE05x_CryptoModeSubType_t

- Defined in [File se05x_enums.h](#)

10.15.3.1.1 Union Documentation

union **SE05x_CryptoModeSubType_t**

#include <se05x_enums.h>

Cyrpto 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

SE05x_PAKEMode_t pakeMode

In case it's pake

uint8_t union_8bit

Accessing 8 bit value for APDUs

10.15.3.2 Union SE05x_I2CM_INS_type_t

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.3.2.1 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.

10.15.4 Functions

10.15.4.1 Function datastore_read_callback

- Defined in [File nxp_iot_agent_datastore.h](#)

10.15.4.1.1 Function Documentation

```
bool datastore_read_callback(pb\_istream\_t *stream, uint8\_t *buf, size\_t count)
```

10.15.4.2 Function DoAPDUTx_s_Case3

- Defined in [File se05x_tlv.h](#)

10.15.4.2.1 Function Documentation

```
smStatus\_t DoAPDUTx_s_Case3(Se05xSession\_t *pSessionCtx, const tlvHeader\_t *hdr, uint8\_t *cmdBuf, size\_t cmdBufLen)
```

10.15.4.3 Function DoAPDUTxRx

- Defined in [File se05x_tlv.h](#)

10.15.4.3.1 Function Documentation

`smStatus_t DoAPDUTxRx(Se05xSession_t *pSessionCtx, uint8_t *cmdBuf, size_t cmdBufLen, uint8_t *rspBuf, size_t *pRspBufLen)`

10.15.4.4 Function DoAPDUTxRx_s_Case2

- Defined in [File se05x_tlv.h](#)

10.15.4.4.1 Function Documentation

`smStatus_t DoAPDUTxRx_s_Case2(Se05xSession_t *pSessionCtx, const tlvHeader_t *hdr, uint8_t *cmdBuf, size_t cmdBufLen, uint8_t *rspBuf, size_t *pRspBufLen)`

10.15.4.5 Function DoAPDUTxRx_s_Case4

- Defined in [File se05x_tlv.h](#)

10.15.4.5.1 Function Documentation

`smStatus_t DoAPDUTxRx_s_Case4(Se05xSession_t *pSessionCtx, const tlvHeader_t *hdr, uint8_t *cmdBuf, size_t cmdBufLen, uint8_t *rspBuf, size_t *pRspBufLen)`

10.15.4.6 Function DoAPDUTxRx_s_Case4_ext

- Defined in [File se05x_tlv.h](#)

10.15.4.6.1 Function Documentation

`smStatus_t DoAPDUTxRx_s_Case4_ext(Se05xSession_t *pSessionCtx, const tlvHeader_t *hdr, uint8_t *cmdBuf, size_t cmdBufLen, uint8_t *rspBuf, size_t *pRspBufLen)`

10.15.4.7 Function encode_responses_callback

- Defined in [File nxp_iot_agent_dispatcher.h](#)

10.15.4.7.1 Function Documentation

`bool encode_responses_callback(pb_ostream_t *ostream, const pb_field_t *field, void *const *arg)`

Encodes the responses callback.

Parameters

- `ostream` – [out] Output stream
- `field` – [in] Input pb field
- `args` – [in] Payload arguments

10.15.4.8 Function encode_responses_from_payload

- Defined in [File nxp_iot_agent_dispatcher.h](#)

10.15.4.8.1 Function Documentation

```
bool encode_responses_from_payload(pb\_ostream\_t *ostream, nxp\_iot\_ResponsePayload *response)
```

Encodes the response from payload.

Parameters

- ostream – [out]** Output stream
- response – [in]** The input payload

10.15.4.9 Function handle_request_payload

- Defined in [File nxp_iot_agent_dispatcher.h](#)

10.15.4.9.1 Function Documentation

```
bool handle_request_payload(pb\_istream\_t *stream, const pb\_field\_t *field, void **arg)
```

Handles the payload request.

Parameters

- stream – [in]** Input stream
- field – [in]** Input pb field
- args – [in]** Payload arguments

10.15.4.10 Function handle_requests

- Defined in [File nxp_iot_agent_dispatcher.h](#)

10.15.4.10.1 Function Documentation

```
bool handle_requests(pb\_istream\_t *stream, const pb\_field\_t *field, void **arg)
```

Handles the request.

Parameters

- stream – [in]** Input stream
- field – [in]** Input pb field
- args – [in]** Payload arguments

10.15.4.11 Function iot_agent_datastore_allocate

- Defined in [File nxp_iot_agent_datastore.h](#)

10.15.4.11.1 Function Documentation

```
iot_agent_status_t iot_agent_datastore_allocate(iot\_agent\_datastore\_t *datastore, size_t len)
```

Allocate memory in a datastore.

Reserve memory in a datastore that can be written to by subsequent writes.

If the memory allocation fails, this function does not return IOT_AGENT_SUCCESS.

10.15.4.12 Function iot_agent_datastore_commit

- Defined in [File nxp_iot_agent_datastore.h](#)

10.15.4.12.1 Function Documentation

`iot_agent_status_t iot_agent_datastore_commit(iot_agent_datastore_t *datastore)`

Commit a transaction to a datastore.

A call to this function indicates that all necessary writes of a transaction are done and the datastore contents can be considered valid.

This is the indication to the datastore to invalidate old contents and from point in time onwards use the data that was updated with the transaction that is committed.

10.15.4.13 Function iot_agent_datastore_encode_datastore_ok_response

- Defined in [File nxp_iot_agent_datastore.h](#)

10.15.4.13.1 Function Documentation

`bool iot_agent_datastore_encode_datastore_ok_response(pb_ostream_t *ostream)`

10.15.4.14 Function iot_agent_datastore_free

- Defined in [File nxp_iot_agent_datastore.h](#)

10.15.4.14.1 Function Documentation

`iot_agent_status_t iot_agent_datastore_free(iot_agent_datastore_t *datastore)`

Destroy the datastore.

Depending on the type of the datastore this is triggering actions on the datastore's context itself by calling the `_destroy()` function of the datastore interface.

This does not free the context of the datastore.

The `datastore` is not usable after a call to `iot_agent_datastore_free`.

10.15.4.15 Function iot_agent_datastore_init

- Defined in [File nxp_iot_agent_datastore.h](#)

10.15.4.15.1 Function Documentation

`iot_agent_status_t iot_agent_datastore_init(iot_agent_datastore_t *datastore, uint32_t identifier, const char *basename, iot_agent_datastore_validator_t validator)`

Initialize a Agent datastore.

The Agent datastore; internally executes all the necessary operations which allows the specific datastore to be initialized

10.15.4.16 Function iot_agent_datastore_read

- Defined in [File nxp_iot_agent_datastore.h](#)

10.15.4.16.1 Function Documentation

`iot_agent_status_t iot_agent_datastore_read(iot_agent_datastore_t *datastore, void *dst, size_t offset, size_t *len)`

Read from a datastore to a buffer in memory.

Starting at position `offset`, `len` bytes are read from the datastore and copied to the memory pointed to by `dst`.

If the datastore does not hold `len` bytes or the read would be out of bounds, only as many bytes as available are read.

`len` is updated to hold the number of bytes that actually were read from the datastore.

No length checks on `dst` are performed, the caller is responsible that the memory location is writeable and can hold `len` bytes.

10.15.4.17 Function iot_agent_datastore_write

- Defined in [File nxp_iot_agent_datastore.h](#)

10.15.4.17.1 Function Documentation

`iot_agent_status_t iot_agent_datastore_write(iot_agent_datastore_t *datastore, size_t offset, const void *src, size_t len)`

Write from a buffer in memory to a datastore.

From `src`, `len` bytes are copied to the datastore. The first byte is written to the position `offset` in the datastore.

Length check on the destination - the datastore - is performed, memory in the datastore needs to be pre-allocated.

If the data does not fit, this function does not return IOT_AGENT_SUCCESS.

10.15.4.18 Function iot_agent_dispatcher

- Defined in [File nxp_iot_agent_dispatcher.h](#)

10.15.4.18.1 Function Documentation

`iot_agent_status_t iot_agent_dispatcher(iot_agent_dispatcher_context_t *dispatcher_context, pb_istream_t *input, pb_ostream_t *output)`

Dispatcher.

Parameters

- `dispatcher_context` – [in] Context for the dispatcher
- `input` – [in] Input stream
- `out` – [in] Output stream

10.15.4.19 Function iot_agent_free_service_descriptor

- Defined in [File nxp_iot_agent.h](#)

10.15.4.19.1 Function Documentation

`void iot_agent_free_service_descriptor(nxp_iot_ServiceDescriptor *service_descriptor)`

Free all FT_POINTER fields of a service descriptor.

When selecting a service, a service descriptor is read from a datastore. A service descriptor can contain fields of variable length (binary data (certificates, etc.) or text (hostname, etc.)). Those fields use dynamically allocated memory. The memory is freed by calling this function.

Parameters

`service_descriptor` – [in] Reference to service descriptor

10.15.4.20 Function `iot_agent_free_update_status_report`

- Defined in [File nxp_iot_agent.h](#)

10.15.4.20.1 Function Documentation

`void iot_agent_free_update_status_report(nxp_iot_UpdateStatusReport *status_report)`

Free all FT_POINTER fields of a update status report.

When a status report is filled during updating a device configuration, it contains pointer fields which use dynamically allocated memory. The memory is freed by calling this function.

Parameters

`status_report` – [in] Reference to status_report

10.15.4.21 Function `iot_agent_get_datastore_by_id`

- Defined in [File nxp_iot_agent.h](#)

10.15.4.21.1 Function Documentation

`iot_agent_status_t iot_agent_get_datastore_by_id(const iot_agent_context_t *ctx, const uint32_t id, iot_agent_datastore_t **datastore)`

Get a reference to a datastore based on its identifier.

Parameters

- `ctx` – [in] The context of the agent.
- `id` – [in] The datastore ID.
- `datastore` – [out] The output datastore.

Return values

`IOT_STATUS_SUCCESS` – A datastore with the `id` was found in the agent's context and a pointer to it is returned in `datastore`.

10.15.4.22 Function `iot_agent_get_datastore_index_by_id`

- Defined in [File nxp_iot_agent.h](#)

10.15.4.22.1 Function Documentation

`iot_agent_status_t iot_agent_get_datastore_index_by_id(const iot_agent_context_t *ctx, const uint32_t id, size_t *index)`

Get the index of a datastore based on its identifier.

Parameters

- `ctx` – [in] The context of the agent.
- `id` – [in] The datastore ID.
- `index` – [out] The output index.

Return values

IOT_STATUS_SUCCESS – A datastore with the `id` was found in the agent's context and its index is returned in `index`.

10.15.4.23 Function `iot_agent_get_endpoint_info`

- Defined in [File nxp_iot_agent.h](#)

10.15.4.23.1 Function Documentation

`bool iot_agent_get_endpoint_info(void *context, void *endpoint_information)`

Get an endpoint information of the endpoint.

Parameters

- `context` – **[in]** Reference to end point context
- `endpoint_information` – **[in]** Reference to end point information

10.15.4.24 Function `iot_agent_get_first_found_object`

- Defined in [File nxp_iot_agent_utils.h](#)

10.15.4.24.1 Function Documentation

`iot_agent_status_t iot_agent_get_first_found_object(iot_agent_keystore_t *keystore, uint32_t *object_ids, size_t num_objects, uint32_t *object_id)`

Maps a given service id to the range of keys that are managed by the EdgeLock 2GO cloud service.

Parameters

- `keystore` – **[in]** The input keystore
- `object_ids` – **[in]** The list of object IDs
- `num_objects` – **[in]** The number of objects
- `object_id` – **[out]** The output object id

Return values

- `IOT_AGENT_SUCCESS` – upon success
- `IOT_AGENT_FAILURE` – upon failure

10.15.4.25 Function `iot_agent_get_keystore_by_id`

- Defined in [File nxp_iot_agent.h](#)

10.15.4.25.1 Function Documentation

`iot_agent_status_t iot_agent_get_keystore_by_id(const iot_agent_context_t *ctx, const uint32_t id, iot_agent_keystore_t **keystore)`

Get a reference to a keystore based on its identifier.

Parameters

- `ctx` – **[in]** The context of the agent.
- `id` – **[in]** The datastore ID.
- `keystore` – **[out]** The output keystore.

Return values

IOT_STATUS_SUCCESS – A keystore with the `id` was found in the agent's context and a pointer to it is returned in `keystore`.

10.15.4.26 Function `iot_agent_get_keystore_index_by_id`

- Defined in [File nxp_iot_agent.h](#)

10.15.4.26.1 Function Documentation

`iot_agent_status_t iot_agent_get_keystore_index_by_id(const iot_agent_context_t *ctx, const uint32_t id, size_t *index)`

Get the index of a keystore based in its identifier.

Parameters

- `ctx` – **[in]** The context of the agent.
- `id` – **[in]** The keystore ID.
- `index` – **[out]** The output index.

Return values

IOT_STATUS_SUCCESS – A keystore with the `id` was found in the agent's context and its index is returned in `index`.

10.15.4.27 Function `iot_agent_get_number_of_services`

- Defined in [File nxp_iot_agent.h](#)

10.15.4.27.1 Function Documentation

`size_t iot_agent_get_number_of_services(const iot_agent_context_t *ctx)`

Returns total number of services of all registered datastores.

Parameters

- `ctx` – **[in]** Context for the `iot_agent`.

Return values

Total – number of services of all registered datastores

10.15.4.28 Function `iot_agent_get_service_descriptor`

- Defined in [File nxp_iot_agent.h](#)

10.15.4.28.1 Function Documentation

`iot_agent_status_t iot_agent_get_service_descriptor(const iot_agent_context_t *ctx, nxp_iot_ServiceDescriptor *service_descriptor)`

Get the service descriptor of the currently selected service.

Parameters

- **ctx – [in]** The context of the agent.
- **service_descriptor – [out]** Structure for holding a service descriptor. Must point to a valid service descriptor object upon invocation. Any FT_POINTER fields in the service descriptor are freed before changing the contents to the service_descriptor of the selected service.

Return values

IOT_AGENT_SUCCESS – Upon success

Pre

The configuration data associated to the context is valid.

Post

In case of success, the service_descriptor structure is filled and needs to be freed after usage by calling [iot_agent_free_service_descriptor](#). In case of failure no freeing is required. Also in case of failures, the contents of service_descriptor are not guaranteed to remain intact.

10.15.4.29 Function iot_agent_handle_request

- Defined in [File nxp_iot_agent.h](#)

10.15.4.29.1 Function Documentation

bool iot_agent_handle_request([pb_istream_t](#) *istream, [pb_ostream_t](#) *ostream, const [pb_field_t](#) *message_type, void *context)

handle request by end point

Parameters

- **istream – [in]** Input stream
- **ostream – [in]** Output stream
- **message_type – [in]** a pointer to the message type fields array
- **context – [in]** End point context

10.15.4.30 Function iot_agent_init

- Defined in [File nxp_iot_agent.h](#)

10.15.4.30.1 Function Documentation

iot_agent_status_t iot_agent_init([iot_agent_context_t](#) *ctx)

Initialize EdgeLock 2GO agent context memory with zeros.

Parameters

ctx – [inout] EdgeLock 2GO agent context

Return values

IOT_AGENT_SUCCESS – Upon success

10.15.4.31 Function iot_agent_init_dispatcher

- Defined in [File nxp_iot_agent.h](#)

10.15.4.31.1 Function Documentation

```
iot_agent_status_t iot_agent_init_dispatcher(iot_agent_dispatcher_context_t *dispatcher_context, iot_agent_context_t *agent_context, nxp_iot_ServiceDescriptor *service_descriptor, nxp_iot_UpdateStatusReport *status_report)
```

Initialize Dispatcher.

Parameters

- dispatcher_context** – **[in]** Context for the dispatcher
- agent_context** – **[in]** Context for the agent
- service_descriptor** – **[in]** The service descriptor containing the connection parameters to connect to the EdgeLock 2GO cloud service.
- status_report** – **[out]** A pointer to a structure that gets filled with a status report after the update is complete. If NULL is given, no status report is created.

10.15.4.32 Function iot_agent_is_service_configuration_data_valid

- Defined in [File nxp_iot_agent.h](#)

10.15.4.32.1 Function Documentation

```
bool iot_agent_is_service_configuration_data_valid(const iot_agent_context_t *ctx)
```

Checks whether service configuration data of all registered datastores is valid.

Parameters

- ctx** – **[in]** Context for the iot_agent.

Return values

- true** – Service configuration data of all registered datastores is valid
- false** – Service configuration data of a registered datastores is invalid

10.15.4.33 Function iot_agent_keystore_close_session

- Defined in [File nxp_iot_agent_keystore.h](#)

10.15.4.33.1 Function Documentation

```
void iot_agent_keystore_close_session(iot_agent_keystore_t *keystore)
```

Close a session/connection to a keystore.

10.15.4.34 Function iot_agent_keystore_file_existence

- Defined in [File nxp_iot_agent_utils.h](#)

10.15.4.34.1 Function Documentation

```
iot_agent_status_t iot_agent_keystore_file_existence(const char *filename, bool forceCreation)
```

Checks existence of a file. If required it forces creation of file.

Parameters

- **filename** – [in] Name of the file
- **forceCreation** – [in] Switch to force creation of the file

Return values

- **IOT_AGENT_SUCCESS** – upon success
- **IOT_AGENT_FAILURE** – upon failure

10.15.4.35 Function iot_agent_keystore_free

- Defined in [File nxp_iot_agent_keystore.h](#)

10.15.4.35.1 Function Documentation

`iot_agent_status_t iot_agent_keystore_free(iot_agent_keystore_t *keystore)`

Destroy the keystore.

Depending on the type of the keystore this is triggering actions on the keystore's context itself by calling the `_destroy()` function of the keystore interface.

The `keystore` is not usable after a call to `iot_agent_keystore_free`.

10.15.4.36 Function iot_agent_keystore_init

- Defined in [File nxp_iot_agent_keystore.h](#)

10.15.4.36.1 Function Documentation

`iot_agent_status_t iot_agent_keystore_init(iot_agent_keystore_t *keystore, uint32_t identifier, iot_agent_platform_context_t *platform_context)`

Initialize an Agent keystore.

The Agent keystore; internally executes all the necessary operations which allows the specific keystore to be initialized as for example opening a SE5x session with the secure element if necessary

If the function does not return `IOT_AGENT_SUCCESS`, it can be assumed that no memory was allocated. The datastore is unusable.

10.15.4.37 Function iot_agent_keystore_open_session

- Defined in [File nxp_iot_agent_keystore.h](#)

10.15.4.37.1 Function Documentation

`iot_agent_status_t iot_agent_keystore_open_session(iot_agent_keystore_t *keystore)`

Open a session/connection to a keystore.

10.15.4.38 Function iot_agent_pad_iso7816d4

- Defined in [File nxp_iot_agent_utils.h](#)

10.15.4.38.1 Function Documentation

```
iot_agent_status_t iot_agent_pad_iso7816d4(uint8_t *data, size_t data_size, size_t unpadded_length, size_t blocksize,  
size_t *padded_length)
```

Append padding bytes to a given message.

Padding is done in-place, meaning the data buffer has to be big enough to hold also the padding bytes.

Parameters

- **data** – **[inout]** Buffer to be unpad
- **data_size** – **[in]** The length of the buffer
- **unpadded_length** – **[in]** The length of the unpadded part
- **blocksize** – **[in]** Size of the block
- **padded_length** – **[inout]** The length of the padded part

Return values

- **IOT_AGENT_SUCCESS** – upon success
- **IOT_AGENT_FAILURE** – upon failure

10.15.4.39 Function iot_agent_print_status_report

- Defined in [File nxp_iot_agent.h](#)

10.15.4.39.1 Function Documentation

```
void iot_agent_print_status_report(const nxp\_iot\_UpdateStatusReport *status_report)
```

Print status report.

It prints the status report of the agent execution.

Parameters

- **status_report** – **[in]** Reference to status_report

10.15.4.40 Function iot_agent_register_datastore

- Defined in [File nxp_iot_agent.h](#)

10.15.4.40.1 Function Documentation

```
iot_agent_status_t iot_agent_register_datastore(iot\_agent\_context\_t *ctx, iot\_agent\_datastore\_t *datastore)
```

Register datastore endpoint.

Note that the ownership for the datastore is not transferred. The caller is responsible that the datastore is freed at the appropriate time.

It is not possible to register two endpoints with the same identifier.

Parameters

- **ctx** – **[in]** Context for the iot_agent.
- **datastore** – **[in]** Datastore that is registered.

Return values

- **IOT_AGENT_SUCCESS** – Upon success

10.15.4.41 Function iot_agent_register_keystore

- Defined in [File nxp_iot_agent.h](#)

10.15.4.41.1 Function Documentation

`iot_agent_status_t iot_agent_register_keystore(iot_agent_context_t *ctx, iot_agent_keystore_t *keystore)`

Register a keystore endpoint.

Note that the ownership for the keystore is not transferred. The caller is responsible that the keystore is freed at the appropriate time.

It is not possible to register two endpoints with the same identifier.

Parameters

- `ctx` – Context for the iot_agent.
- `keystore` – [in] Keystore that is registered.

Return values

`IOT_AGENT_SUCCESS` – Upon success

10.15.4.42 Function iot_agent_select_service_by_id

- Defined in [File nxp_iot_agent.h](#)

10.15.4.42.1 Function Documentation

`iot_agent_status_t iot_agent_select_service_by_id(iot_agent_context_t *ctx, uint64_t service_id, nxp_iot_ServiceDescriptor *service_descriptor)`

Select service by given ID.

See Also:

[*iot_agent_is_service_configuration_data_valid*](#)

Parameters

- `ctx` – [in] Context for the iot_agent.
- `service_id` – [in] ID of the service
- `service_descriptor` – [out] Structure for holding a service descriptor. Must point to a valid service descriptor object upon invocation. Any FT_POINTER fields in the service descriptor are freed before changing the contents to the service_descriptor of the selected service.

Return values

`IOT_AGENT_SUCCESS` – Upon success

Pre

The configuration data associated to the context is valid.

Post

In case of success, the service_descriptor structure is filled and needs to be freed after usage by calling [*iot_agent_free_service_descriptor*](#). In case of failure no freeing is required. Also in case of failures, the contents of service_descriptor are not guaranteed to remain intact.

10.15.4.43 Function iot_agent_select_service_by_index

- Defined in [File nxp_iot_agent.h](#)

10.15.4.43.1 Function Documentation

`iot_agent_status_t iot_agent_select_service_by_index(iot_agent_context_t *ctx, size_t index, nxp_iot_ServiceDescriptor *service_descriptor)`

Select service by given index.

See Also:

[iot_agent_is_service_configuration_data_valid](#)

Parameters

- `ctx – [in]` Context for the iot_agent.
- `index – [in]` Index of the service
- `service_descriptor – [out]` Structure for holding a service descriptor. Must point to a valid service descriptor object upon invocation. Any FT_POINTER fields in the service descriptor are freed before changing the contents to the service_descriptor of the selected service.

Return values

`IOT_AGENT_SUCCESS` – Upon success

Pre

The configuration data associated to the context is valid.

Post

In case of success, the service_descriptor structure is filled and needs to be freed after usage by calling [iot_agent_free_service_descriptor](#). In case of failure no freeing is required. Also in case of failures, the contents of service_descriptor are not guaranteed to remain intact.

10.15.4.44 Function iot_agent_service_free_service_descriptor

- Defined in [File nxp_iot_agent_service.h](#)

10.15.4.44.1 Function Documentation

`void iot_agent_service_free_service_descriptor(nxp_iot_ServiceDescriptor *service_descriptor)`

Free all FT_POINTER fields of a service descriptor.

When selecting a service, a service descriptor is read from a datastore. A service descriptor can contain fields of variable length (binary data (certificates, etc.) or text (hostname, etc.)). Those fields use dynamically allocated memory. The memory is freed by calling this function.

Parameters

`service_descriptor – [in]` Reference to service descriptor

10.15.4.45 Function iot_agent_service_get_number_of_services

- Defined in [File nxp_iot_agent_service.h](#)

10.15.4.45.1 Function Documentation

`size_t iot_agent_service_get_number_of_services(const iot_agent_datastore_t *ctx)`

Get the number of services that are available.

Parameters

`ctx – [in]` Context for the iot_agent.

Returns

The number of services that are available.

10.15.4.46 Function `iot_agent_service_get_protocol_of_service_as_string`

- Defined in [File nxp_iot_agent_service.h](#)

10.15.4.46.1 Function Documentation

`iot_agent_status_t iot_agent_service_get_protocol_of_service_as_string(const nxp_iot_ServiceDescriptor *service_descriptor, const char **buffer)`

Get a textual description of the protocol.

Protocol strings are internal constant c-strings. This function returns a pointer to such a string, ownership remains with the iot_agent.

Parameters

- `service_descriptor – [in]` The service descriptor to get the protocol for.
- `buffer – [out]` A pointer to a c-string that is changed to point to the textual representation of the protocol.

Return values

- `IOT_AGENT_SUCCESS` – The service type could be resolved.
- `IOT_AGENT_FAILURE` – The service type is invalid.

10.15.4.47 Function `iot_agent_service_get_service_descriptor_of_service`

- Defined in [File nxp_iot_agent_service.h](#)

10.15.4.47.1 Function Documentation

`iot_agent_status_t iot_agent_service_get_service_descriptor_of_service(const iot_agent_datastore_t *datastore, size_t offset, nxp_iot_ServiceDescriptor *service_descriptor)`

Get the service descriptor of a service specified by offset.

See Also:

[`iot_agent_is_service_configuration_data_valid`](#)

Parameters

- `ctx – [in]` The datastore to query for the service.
- `offset – [in]` The offset of the service - the memory location in the given datastore.
- `service_descriptor – [out]` Structure for holding a service descriptor. Must point to a valid service descriptor object upon invocation. Any FT_POINTER fields in the service descriptor are freed before changing the contents to the service_descriptor of the selected service.

Return values

`IOT_AGENT_SUCCESS` – Upon success

Pre

The configuration data associated to the context is valid.

Post

In case of success, the `service_descriptor` structure is filled and needs to be freed after usage by calling `iot_agent_free_service_descriptor`. In case of failure no freeing is required. Also in case of failures, the contents of `service_descriptor` are not guaranteed to remain intact.

10.15.4.48 Function `iot_agent_service_get_service_offset_by_id`

- Defined in [File nxp_iot_agent_service.h](#)

10.15.4.48.1 Function Documentation

`iot_agent_status_t iot_agent_service_get_service_offset_by_id(const iot_agent_datastore_t *datastore, uint64_t service_id, size_t *offset, nxp_iot_ServiceDescriptor *service_descriptor)`

Get the offset and the service descriptor of a service specified by `service_id`.

See Also:

[`iot_agent_is_service_configuration_data_valid`](#)

Parameters

- `datastore` – **[in]** The datastore to query for the service.
- `service_id` – **[in]** ID of the service
- `offset` – **[out]** The offset of the service descriptor within the datastore.
- `service_descriptor` – **[out]** Structure for holding a service descriptor. Must point to a valid service descriptor object upon invocation. Any FT_POINTER fields in the service descriptor are freed before changing the contents to the `service_descriptor` of the selected service.

Return values

`IOT_AGENT_SUCCESS` – Upon success

Pre

The configuration data associated to the context is valid.

Post

In case of success, the `service_descriptor` structure is filled and needs to be freed after usage by calling `iot_agent_free_service_descriptor`. In case of failure no freeing is required. Also in case of failures, the contents of `service_descriptor` are not guaranteed to remain intact.

10.15.4.49 Function `iot_agent_service_get_service_offset_by_index`

- Defined in [File nxp_iot_agent_service.h](#)

10.15.4.49.1 Function Documentation

`iot_agent_status_t iot_agent_service_get_service_offset_by_index(const iot_agent_datastore_t *datastore, size_t index, size_t *offset, nxp_iot_ServiceDescriptor *service_descriptor)`

Get the offset and the service descriptor of a service specified by index.

See Also:

[*iot_agent_is_service_configuration_data_valid*](#)

Parameters

- **ctx – [in]** The datastore to query for the service.
- **index – [in]** The index of the service (within the given datastore).
- **service_descriptor – [out]** Structure for holding a service descriptor. Must point to a valid service descriptor object upon invocation. Any FT_POINTER fields in the service descriptor are freed before changing the contents to the service_descriptor of the selected service.

Return values

IOT_AGENT_SUCCESS – Upon success

Pre

The configuration data associated to the context is valid.

Post

In case of success, the service_descriptor structure is filled and needs to be freed after usage by calling [*iot_agent_free_service_descriptor*](#). In case of failure no freeing is required. Also in case of failures, the contents of service_descriptor are not guaranteed to remain intact.

10.15.4.50 Function *iot_agent_service_get_service_type_as_string*

- Defined in [*File nxp_iot_agent_service.h*](#)

10.15.4.50.1 Function Documentation

*iot_agent_status_t iot_agent_service_get_service_type_as_string(const nxp_iot_ServiceDescriptor *service_descriptor, const char **buffer)*

Get a textual description of the service type.

Service type strings are internal constant c-strings. This function returns a pointer to such a string, ownership remains with the iot_agent.

Parameters

- **service_descriptor – [in]** The service descriptor to get the type for.
- **buffer – [out]** A pointer to a c-string that is changed to point to the textual representation of the service type.

Return values

- **IOT_AGENT_SUCCESS** – The service type could be resolved.
- **IOT_AGENT_FAILURE** – The service type is invalid.

10.15.4.51 Function *iot_agent_service_is_configuration_data_valid*

- Defined in [*File nxp_iot_agent_service.h*](#)

10.15.4.51.1 Function Documentation

*bool iot_agent_service_is_configuration_data_valid(const iot_agent_datastore_t *ctx)*

Checks service configuration data of all registered datastores are valid.

Parameters

ctx – [in] Context for the iot_agent.

Return values

- **true** – Service configuration data of all registered datastores are valid
- **false** – Service configuration data of a registered datastores is invalid

10.15.4.52 Function iot_agent_set_edgelock2go_datastore

- Defined in [File nxp_iot_agent.h](#)

10.15.4.52.1 Function Documentation

`iot_agent_status_t iot_agent_set_edgelock2go_datastore(iot_agent_context_t *ctx, iot_agent_datastore_t *datastore)`

Set the datastore that is used to hold the information to connect to the EdgeLock 2GO cloud service.

Note that the ownership for the datastore is not transferred. The caller is responsible that the datastore is freed at the appropriate time.

Parameters

- **ctx** – [in] Context for the iot_agent.
- **datastore** – [in] Datastore to be set.

10.15.4.53 Function iot_agent_update_device_configuration

- Defined in [File nxp_iot_agent.h](#)

10.15.4.53.1 Function Documentation

`iot_agent_status_t iot_agent_update_device_configuration(iot_agent_context_t *agent_context, nxp_iot_UpdateStatusReport *status_report)`

Update device configuration Reach out to EdgeLock 2GO cloud service for checking and (if applicable) fetching configuration updates for the device.

Parameters

- **agent_context** – [in] Context for the iot_agent.
- **status_report** – [out] Provides a more detailed view on the operations performed during the update and its outcomes. If the argument is NULL, no detailed status is reported.

Return values

- **IOT_AGENT_SUCCESS** – Upon success
- **IOT_AGENT_FAILURE** – Upon failure

Post

In case of success, the status_report structure is filled using dynamically allocated fields and needs to be freed after usage by calling #iot_agent_free_status_report.

10.15.4.54 Function iot_agent_update_device_configuration_from_constants

- Defined in [File nxp_iot_agent.h](#)

10.15.4.54.1 Function Documentation

`iot_agent_status_t iot_agent_update_device_configuration_from_constants(iot_agent_context_t *agent_context, nxp_iot_UpdateStatusReport *status_report)`

Update device configuration from constants.

Reach out to EdgeLock 2GO cloud service for checking and (if applicable) fetching configuration updates for the device.

The connection details (hostname/port/server root certificates, etc.) are taken from the configuration constants in `nxp_iot_agent_config.h`.

It is necessary that an sss keystore that contains credentials (client key and client certificate) for connecting to the EdgeLock 2GO cloud service. The object ids to those credentials are settable via function arguments.

Parameters

- `ctx – [in]` Context for the `iot_agent`.
- `status_report – [out]` Provides a more detailed view on the operations performed during the update and its outcomes. If the argument is NULL, no detailed status is reported.

Return values

- `IOT_AGENT_SUCCESS` – Upon success
- `IOT_AGENT_FAILURE` – Upon failure

Post

In case of success, the `status_report` structure is filled using dynamically allocated fields and needs to be freed after usage by calling `#iot_agent_free_status_report`.

10.15.4.55 Function `iot_agent_update_device_configuration_from_datastore`

- Defined in [File nxp_iot_agent.h](#)

10.15.4.55.1 Function Documentation

`iot_agent_status_t iot_agent_update_device_configuration_from_datastore(iot_agent_context_t *agent_context, iot_agent_datastore_t *datastore, nxp_iot_UpdateStatusReport *status_report)`

Update device configuration from datastore.

Reach out to EdgeLock 2GO cloud service for checking and (if applicable) fetching configuration updates for the device.

The connection details (hostname/port/server root certificates, etc.) are taken from the datastore passed as input parameter

It is necessary that an sss keystore that contains credentials (client key and client certificate) for connecting to the EdgeLock 2GO cloud service. The object ids to those credentials are settable via function arguments.

Parameters

- `ctx – [in]` Context for the `iot_agent`.
- `datastore – [in]` Datastore including the connection parameters
- `status_report – [out]` Provides a more detailed view on the operations performed during the update and its outcomes. If the argument is NULL, no detailed status is reported.

Return values

- `IOT_AGENT_SUCCESS` – Upon success
- `IOT_AGENT_FAILURE` – Upon failure

Post

In case of success, the status_report structure is filled using dynamically allocated fields and needs to be freed after usage by calling #iot_agent_free_status_report.

10.15.4.56 Function iot_agent_update_device_configuration_from_service_descriptor

- Defined in [File nxp_iot_agent.h](#)

10.15.4.56.1 Function Documentation

iot_agent_status_t iot_agent_update_device_configuration_from_service_descriptor([iot_agent_context_t](#) *agent_context, [nxp_iot_ServiceDescriptor](#) *service_descriptor, [nxp_iot_UpdateStatusReport](#) *status_report)

Update device configuration from service descriptor.

Reach out to EdgeLock 2GO cloud service for checking and (if applicable) fetching configuration updates for the device.

The connection details (hostname/port/server root certificates, etc.) are taken from the service descriptor passed as input parameter

It is necessary that an sss keystore that contains credentials (client key and client certificate) for connecting to the EdgeLock 2GO cloud service. The object ids to those credentials are settable via function arguments.

Parameters

- ctx – [in]** Context for the iot_agent.
- service_descriptor – [in]** Service descriptor including the connection parameters
- status_report – [out]** Provides a more detailed view on the operations performed during the update and its outcomes. If the argument is NULL, no detailed status is reported.

Return values

- IOT_AGENT_SUCCESS** – Upon success
- IOT_AGENT_FAILURE** – Upon failure

Post

In case of success, the status_report structure is filled using dynamically allocated fields and needs to be freed after usage by calling #iot_agent_free_status_report.

10.15.4.57 Function iot_agent_utils_configure_edgelock2go_datastore

- Defined in [File nxp_iot_agent_utils.h](#)

10.15.4.57.1 Function Documentation

Warning:

doxygenfunction: Unable to resolve function “iot_agent_utils_configure_edgelock2go_datastore” with arguments ([iot_agent_keystore_t](#)*, [iot_agent_datastore_t](#)*, [int](#), [const char](#)*) in doxygen xml output for project “simw” from directory: ../../_docs_build_simw/xml. Potential matches:

```
- iot_agent_status_t  
  iot_agent_utils_configure_edgelock2go_datastore(iot\_agent\_keystore\_t *keystore,  
  iot\_agent\_datastore\_t *datastore, int argc, const char *argv[])
```

10.15.4.58 Function iot_agent_utils_convert_service2key_id

- Defined in [File nxp_iot_agent_utils.h](#)

10.15.4.58.1 Function Documentation

`iot_agent_status_t iot_agent_utils_convert_service2key_id(uint64_t service_id, uint32_t *key_id_ptr)`

Converts the service ID to key id.

Parameters

`service_id – [in]` Service ID @ param[out] `key_id_ptr` Key ID

Return values

- `IOT_AGENT_SUCCESS` – upon success
- `IOT_AGENT_FAILURE` – upon failure

10.15.4.59 Function iot_agent_utils_create_self_signed_edgelock2go_certificate

- Defined in [File nxp_iot_agent_utils.h](#)

10.15.4.59.1 Function Documentation

`iot_agent_status_t iot_agent_utils_create_self_signed_edgelock2go_certificate(iot_agent_keystore_t *keystore, uint8_t *certificate_buffer, size_t *certificate_buffer_size)`

Creates a self signed certificate.

Parameters

- `keystore – [in]` The input keystore
- `certificate_buffer – [out]` Buffer including the certificate
- `certificate_buffer_size – [inout]` the length of the certificate

Return values

- `IOT_AGENT_SUCCESS` – upon success
- `IOT_AGENT_FAILURE` – upon failure

10.15.4.60 Function iot_agent_utils_gen_key_ref

- Defined in [File nxp_iot_agent_utils.h](#)

10.15.4.60.1 Function Documentation

`iot_agent_status_t iot_agent_utils_gen_key_ref(const iot_agent_keystore_t *keystore, uint32_t key_id, iot_agent_keystore_key_ref_t *gen_key_ref)`

Generates the key reference. The function is used when OpenSSL crypto is selected to generate the Open SSL key reference using the key stored in the keystore. The key reference provided as output depends on the used OpenSSL version.

- Version < 3.x.x: EVP_PKEY
- Version >= 3.x.x: OSSL_PARAM

Parameters

- `keystore – [in]` The input keystore
- `key_id – [in]` ID of the key
- `gen_key_ref – [out]` The generated key reference

Return values

- **IOT_AGENT_SUCCESS** – upon success
- **IOT_AGENT_FAILURE** – upon failure

10.15.4.61 Function iot_agent_utils_get_certificate_common_name

- Defined in [File nxp_iot_agent_utils.h](#)

10.15.4.61.1 Function Documentation

`iot_agent_status_t iot_agent_utils_get_certificate_common_name(iot_agent_context_t *ctx, const nxp_iot_ServiceDescriptor *service_descriptor, char *common_name, size_t max_size)`

Gets the common name from the client certificate.

Parameters

- **ctx** – **[in]** Context for the iot_agent
- **service_descriptor** – **[in]** Descriptor with service data
- **common_name** – **[inout]** Common name string
- **max_size** – **[in]** Maximum size

Return values

- **IOT_AGENT_SUCCESS** – upon success
- **IOT_AGENT_FAILURE** – upon failure

10.15.4.62 Function iot_agent_utils_get_certificate_from_keystore

- Defined in [File nxp_iot_agent_utils.h](#)

10.15.4.62.1 Function Documentation

`iot_agent_status_t iot_agent_utils_get_certificate_from_keystore(iot_agent_keystore_t *keystore, uint32_t certificate_id, uint8_t *cert, size_t *cert_len)`

Read a certificate from a keystore.

Parameters

- **keystore** – **[in]** The input keystore
- **certificate_id** – **[in]** ID of the certificate
- **cert** – **[out]** Buffer including the certificate
- **cert_len** – **[inout]** the length of the certificate

Return values

- **IOT_AGENT_SUCCESS** – upon success
- **IOT_AGENT_FAILURE** – upon failure

10.15.4.63 Function iot_agent_utils_get_device_id

- Defined in [File nxp_iot_agent_utils.h](#)

10.15.4.63.1 Function Documentation

`iot_agent_status_t iot_agent_utils_get_device_id(uint8_t *buffer, size_t *len)`

Retrieve the UID of a device.

Parameters

- **buffer** – **[out]** The buffer including the UUID
- **len** – **[inout]** The length of the buffer

Return values

- **IOT_AGENT_SUCCESS** – upon success
- **IOT_AGENT_FAILURE** – upon failure

10.15.4.64 Function `iot_agent_utils_get_edgelock2go_certificate_id`

- Defined in [File nxp_iot_agent_utils.h](#)

10.15.4.64.1 Function Documentation

`iot_agent_status_t iot_agent_utils_get_edgelock2go_certificate_id(iot_agent_keystore_t *keystore, uint32_t *object_id)`

Checks whether a keystore contains the object with the defined certificate to use for authenticating at the EdgeLock 2GO cloud service.

The keystore might contain keys using ECC and/or RSA. If available, it will return the object id of the ECC certificate, otherwise the object id of the RSA certificate.

When neither is found or in case of other issues with the keystore, the function does not return IOT_AGENT_SUCCESS.

Parameters

- **keystore** – **[in]** The input keystore
- **object_id** – **[out]** ID of the certificate

Return values

- **IOT_AGENT_SUCCESS** – upon success
- **IOT_AGENT_FAILURE** – upon failure

10.15.4.65 Function `iot_agent_utils_get_edgelock2go_key_id`

- Defined in [File nxp_iot_agent_utils.h](#)

10.15.4.65.1 Function Documentation

`iot_agent_status_t iot_agent_utils_get_edgelock2go_key_id(iot_agent_keystore_t *keystore, uint32_t *object_id)`

Checks whether a keystore contains the object with the defined key to use for authenticating at the EdgeLock 2GO cloud service.

The keystore might contain keys using ECC and/or RSA. If available, it will return the object id of the ECC key, otherwise the object id of the RSA key.

When neither is found or in case of other issues with the keystore, the function does not return IOT_AGENT_SUCCESS.

Parameters

- **keystore** – **[in]** The input keystore
- **object_id** – **[out]** ID of the key

Return values

- **IOT_AGENT_SUCCESS** – upon success
- **IOT_AGENT_FAILURE** – upon failure

10.15.4.66 Function iot_agent_utils_get_keystore_from_service_descriptor

- Defined in [File nxp_iot_agent_utils.h](#)

10.15.4.66.1 Function Documentation

`iot_agent_status_t iot_agent_utils_get_keystore_from_service_descriptor(const iot_agent_context_t *ctx, const nxp_iot_ServiceDescriptor *service_descriptor, iot_agent_keystore_t **keystore)`

Gets the keystore from service descriptor.

Parameters

- `ctx` – **[in]** Agent context
- `service_descriptor` – **[in]** Input service descriptor
- `keystore` – **[out]** The output keystore

Return values

- `IOT_AGENT_SUCCESS` – upon success
- `IOT_AGENT_FAILURE` – upon failure

10.15.4.67 Function iot_agent_utils_write_certificate_pem

- Defined in [File nxp_iot_agent_utils.h](#)

10.15.4.67.1 Function Documentation

`iot_agent_status_t iot_agent_utils_write_certificate_pem(uint8_t *buffer, size_t len, const char *filename)`

Create a PEM file containing a certificate chain.

Parameters

- `buffer` – **[in]** Buffer containing the encoded certificate
- `len` – **[in]** length of the buffer
- `filename` – **[in]** Name of the PEM file to be created

Return values

- `IOT_AGENT_SUCCESS` – upon success
- `IOT_AGENT_ERROR_CRYPTO_ENGINE_FAILED` – upon crypto operation failure
- `IOT_AGENT_ERROR_FILE_SYSTEM` – upon failure while creating or writing certificate file
- `IOT_AGENT_FAILURE` – upon failure

10.15.4.68 Function iot_agent_utils_write_certificate_pem_cos_over_rtp

- Defined in [File nxp_iot_agent_utils.h](#)

10.15.4.68.1 Function Documentation

`iot_agent_status_t iot_agent_utils_write_certificate_pem_cos_over_rtp(iot_agent_context_t *ctx, const nxp_iot_ServiceDescriptor *service_descriptor, const char *filename)`

Create a PEM file containing service certificate stored in keystore.

Parameters

- **ctx** – [in] Context for the iot_agent
- **service_descriptor** – [in] Descriptor with service data
- **filename** – [in] Name of the PEM file to be created

Return values

- **IOT_AGENT_SUCCESS** – upon success
- **IOT_AGENT_FAILURE** – upon failure

10.15.4.69 Function iot_agent_utils_write_certificate_pem_from_keystore

- Defined in [File nxp_iot_agent_utils.h](#)

10.15.4.69.1 Function Documentation

`iot_agent_status_t iot_agent_utils_write_certificate_pem_from_keystore(iot_agent_keystore_t *keyStore, const uint32_t certid, const char *filename)`

Create a PEM file containing a certificate.

Parameters

- **keyStore** – [in] Key store context
- **certid** – [in] Certificate id
- **filename** – [in] Name of the PEM file to be created

Return values

- **IOT_AGENT_SUCCESS** – upon success
- **IOT_AGENT_ERROR_CRYPTO_ENGINE_FAILED** – upon crypto operation failure
- **IOT_AGENT_ERROR_FILE_SYSTEM** – upon failure while creating or writing certificate file
- **IOT_AGENT_FAILURE** – upon failure

10.15.4.70 Function iot_agent_utils_write_edgelock2go_datastore

- Defined in [File nxp_iot_agent_utils.h](#)

10.15.4.70.1 Function Documentation

`iot_agent_status_t iot_agent_utils_write_edgelock2go_datastore(iot_agent_keystore_t *keystore, iot_agent_datastore_t *datastore, const char *hostname, uint32_t port, const pb_bytes_array_t *trusted_root_ca_certificates, const pb_bytes_array_t *client_certificate)`

Assemble a service descriptor for the connection to EdgeLock 2GO cloud service and write it to a datastore.

It is assumed that keys (either ECC or RSA) for authentication are stored in keystore.

If `client_certificate` is NULL, it is assumed that the client certificate is stored in keystore as well - in this case the resulting datastore contents will contain a reference to the certificate.

If `client_certificate` is not NULL, it shall point to a client certificate, which will be copied into the service descriptor.

Parameters

- **keystore** – [in] The input keystore
- **datastore** – [out] The datastore where to write
- **hostname** – [in] The hostname URL
- **port** – [in] The port number
- **trusted_root_ca_certificates** – [in] Buffer including the root certificates
- **client_certificate** – [in] Buffer including client certificate

Return values

- **IOT_AGENT_SUCCESS** – upon success
- **IOT_AGENT_FAILURE** – upon failure

10.15.4.71 Function iot_agent_utils_write_key_ref_pem

- Defined in [File nxp_iot_agent_utils.h](#)

10.15.4.71.1 Function Documentation

`iot_agent_status_t iot_agent_utils_write_key_ref_pem(iot_agent_keystore_key_ref_t *gen_key_ref, const char *filename)`

Writes the key reference to PEM. The function is used when OpenSSL crypto is selected to write the key reference generated by the Open SSL to a file. The key reference provided as input depends on the used OpenSSL version.

- Version < 3.x.x: EVP_PKEY
- Version >= 3.x.x: OSSL_PARAM

Parameters

- **gen_key_ref** – [in] The key reference to be written
- **filename** – [in] name of file

Return values

- **IOT_AGENT_SUCCESS** – upon success
- **IOT_AGENT_FAILURE** – upon failure

10.15.4.72 Function iot_agent_utils_write_key_ref_pem_from_keystore

- Defined in [File nxp_iot_agent_utils.h](#)

10.15.4.72.1 Function Documentation

`iot_agent_status_t iot_agent_utils_write_key_ref_pem_from_keystore(iot_agent_keystore_t *keyStore, const uint32_t keyid, const char *filename)`

Create a PEM file containing a key.

Parameters

- **keyStore** – [in] Key store context
- **keyid** – [in] Key id
- **filename** – [in] Name of the PEM file to be created

Return values

- **IOT_AGENT_SUCCESS** – upon success
- **IOT_AGENT_ERROR_CRYPTO_ENGINE_FAILED** – upon crypto operation failure
- **IOT_AGENT_ERROR_FILE_SYSTEM** – upon failure while creating or writing certificate file
- **IOT_AGENT_FAILURE** – upon failure

10.15.4.73 Function `iot_agent_utils_write_key_ref_service_pem`

- Defined in [File nxp_iot_agent_utils.h](#)

10.15.4.73.1 Function Documentation

`iot_agent_status_t iot_agent_utils_write_key_ref_service_pem(const iot_agent_context_t *ctx, const char *filename)`

Create a PEM file containing a key reference to the key of the currently selected service.

Parameters

- **ctx** – **[in]** Context for the iot_agent
- **filename** – **[in]** Name of the PEM file to be created

Return values

- **IOT_AGENT_SUCCESS** – upon success
- **IOT_AGENT_FAILURE** – upon failure

Pre

A service needs to be selected before calling this function.

10.15.4.74 Function `main`

- Defined in [File ex_sss_main_inc.h](#)

10.15.4.74.1 Function Documentation**Warning:**

doxygenfunction: Unable to resolve function “main” with arguments (int, const char) in doxygen xml output for project “simw” from directory: ../../../../../docs_build_simw/xml. Potential matches:*

```
- int main(int argc, const char *argv[])
```

10.15.4.75 Function `mbedtls_ecp_keypair_free_o`

- Defined in [File sss_mbedtls.h](#)

10.15.4.75.1 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.

10.15.4.76 Function mbedtls_ecp_tls_read_group_o

- Defined in [File sss_mbedtls.h](#)

10.15.4.76.1 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
```

10.15.4.77 Function nxp_iot_agent_unpad_iso7816d4

- Defined in [File nxp_iot_agent_utils.h](#)

10.15.4.77.1 Function Documentation

```
iot_agent_status_t nxp_iot_agent_unpad_iso7816d4(uint8_t *data, size_t *data_size)
```

Updates the length in order to remove padded data.

Parameters

- data – [inout]** Buffer to be unpad
- data_size – [inout]** The length of the buffer

Return values

- IOT_AGENT_SUCCESS** – upon success
- IOT_AGENT_FAILURE** – upon failure

10.15.4.78 Function Se05x_API_AeadCCMFinal

- Defined in [File se05x_04_xx_APDU_apis.h](#)

10.15.4.78.1 Function Documentation

```
smStatus_t Se05x_API_AeadCCMFinal(pSe05xSession_t session_ctx, SE05x_CryptoObjectID_t cryptoObjectID, uint8_t
*pOutputData, size_t *pOutputLen, uint8_t *pTag, size_t *pTagLen, const SE05x_Cipher_Oper_t operation)
```

Se05x_API_AeadCCMFinal

Finish a sequence of AES_CCM AEAD operations.

Command to Applet

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	SE05x_INS_t
P1	P1_AEAD	See SE05x_P1_t
P2	P2_FINAL	See SE05x_P2_t
Lc	#(Payload)	
Payload	TLV[TAG_2]	2-byte Crypto Object identifier
TLV[TAG_6]	Byte array containing tag to verify [Conditional] When the mode is decrypt and verify (i.e. AEADInit has been called with P2 = P2_ DECRYPT).	

Field	Value	Description
Le	0x00	Expected returned data.

R-APDU Body

Value	Description
TLV[TAG_1]	Output data
TLV[TAG_2]	Byte array containing tag (if P2 = P2_ENCRYPT) or byte array containing Result (if P2 = P2_DECRYPT)

R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

Parameters

- **session_ctx** – **[in]** The session context
- **cryptoObjectID** – **[in]** The crypto object id
- **pOutputData** – **[out]** The output data
- **pOutputLen** – **[out]** The output length
- **pTag** – The tag
- **pTagLen** – The tag length
- **operation** – **[in]** The operation

Returns

The sm status.

10.15.4.79 Function Se05x_API_AeadCCMInit

- Defined in [File se05x_04_xx_APDU_apis.h](#)

10.15.4.79.1 Function Documentation

smStatus_t Se05x_API_AeadCCMInit(pSe05xSession_t session_ctx, uint32_t objectID, SE05x_CipherMode_t cipherMode, SE05x_CryptoObjectID_t cryptoObjectID, uint8_t *pIV, size_t IVLen, size_t aadLen, size_t payloadLen, size_t tagLen, const SE05x_Cipher_Oper_t operation)

Se05x_API_AeadCCMInit

Initialize an authentication encryption or decryption with associated data. The Crypto Object keeps the state of the AEAD operation until it's finalized or deleted. Once the AEADFinal function is executed successfully, the Crypto Object state returns to the state immediately after the previous AEADInit function.AEAD in CCM mode.

Command to Applet

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	SE05x_INS_t
P1	P1_AEAD	See SE05x_P1_t
P2	P2_ENCRYPT or P2_DECRYPT	See SE05x_P2_t
Lc	#(Payload)	

Field	Value	Description
Payload	TLV[TAG_1]	4-byte identifier of the AESKey Secure object.
	TLV[TAG_2]	2-byte Crypto Object identifier
	TLV[TAG_5]	Byte array containing the initialization vector [12 bytes until 60 bytes] or a 2-byte value containing the initialization vector length, depending on the AEADMode of the Crypto Object.
	TLV[TAG_6]	Byte array containing 2-byte AAD length. [Conditional: needed if AEADMode equals AES_CCM]
	TLV[TAG_7]	Byte array containing 2-byte message length. [Conditional: needed if AEADMode equals AES_CCM]
	TLV[TAG_8]	Byte array containing 2-byte tag size. [Conditional: needed if AEADMode equals AES_CCM].
Le	•	

R-APDU Body

NA R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

Parameters

- **session_ctx – [in]** The session context
- **objectID – [in]** The object id
- **cipherMode – [in]** The cipher mode
- **cryptoObjectID – [in]** The crypto object id
- **pIV – [in] { parameter_description }**
- **IVLen – [in]** The iv length
- **aadLen – [in]** The aad length
- **payloadLen – [in]** The payloadLen length
- **tagLen – [in]** The tag length
- **operation – [in]** The operation

Returns

The sm status.

10.15.4.80 Function Se05x_API_AeadCCMLastUpdate

- Defined in [File se05x_04_xx_APDU_apis.h](#)

10.15.4.80.1 Function Documentation

smStatus_t Se05x_API_AeadCCMLastUpdate(pSe05xSession_t session_ctx, SE05x_CryptoObjectID_t cryptoObjectID, const uint8_t *pInputData, size_t inputDataLen)

Se05x_API_AeadCCMLastUpdate.

Command to Applet

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	SE05x_INS_t
P1	P1_AEAD	See SE05x_P1_t
P2	P2_UPDATE	See SE05x_P2_t
Lc	#(Payload)	
Payload	TLV[TAG_2]	2-byte Crypto Object identifier
	TLV[TAG_3]	Byte array containing input data [Conditional: only when TLV[TAG_4] is not present] [Optional]
Le	0x00	Expecting returned data.

*R-APDU Body*NA *R-APDU Trailer*

SW	Description
SW_NO_ERROR	The command is handled successfully.

*R-APDU Trailer***Parameters**

- **session_ctx – [in]** The session context
- **cryptoObjectID – [in]** The crypto object id
- **pInputData – [in]** The input data
- **inputDataLen – [in]** The input data length

Returns

The sm status.

10.15.4.81 Function Se05x_API_AeadFinal

- Defined in [File se05x_04_xx_APDU_apis.h](#)

10.15.4.81.1 Function Documentation

smStatus_t Se05x_API_AeadFinal(*pSe05xSession_t* session_ctx, *SE05x_CryptoObjectID_t* cryptoObjectID, *uint8_t* *tag, *size_t* *tagLen, const *SE05x_Cipher_Open_t* operation)

Se05x_API_AeadFinal

Finish a sequence of AEAD operations. The AEADFinal command provides the computed GMAC or indicates whether the GMAC is correct depending on the P2 parameters passed during AEADInit. The length of the GMAC is always 16 bytes when P2 equals P2_ENCRYPT. When P2 equals P2_DECRYPT, the minimum tag length to pass is 4 bytes.

Command to Applet

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	SE05x_INS_t
P1	P1_AEAD	See SE05x_P1_t

Field	Value	Description
P2	P2_FINAL	See SE05x_P2_t
Lc	#(Payload)	
Payload	TLV[TAG_2]	2-byte Crypto Object identifier
TLV[TAG_6]	Byte array containing tag to verify [Conditional] When the mode is decrypt and verify (i.e. AEADInit has been called with P2 = P2_ DECRYPT).	
Le	0x00	Expected returned data.

R-APDU Body

Value	Description
TLV[TAG_2]	Byte array containing tag (if P2 = P2_ENCRYPT) or byte array containing Result (if P2 = P2_DECRYPT)

R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

Parameters

- **session_ctx – [in]** The session context
- **cryptoObjectID – [in]** The crypto object id
- **tag –** The tag
- **tagLen –** The tag length
- **operation – [in]** The operation

Returns

The sm status.

10.15.4.82 Function Se05x_API_AeadInit

- Defined in [File se05x_04_xx_APDU_apis.h](#)

10.15.4.82.1 Function Documentation

smStatus_t Se05x_API_AeadInit(*pSe05xSession_t* session_ctx, *uint32_t* objectID, *SE05x_CipherMode_t* cipherMode, *SE05x_CryptoObjectID_t* cryptoObjectID, *uint8_t* *pIV, *size_t* IVLen, const *SE05x_Cipher_Oper_t* operation)

Se05x_API_AeadInit

Initialize an authentication encryption or decryption with associated data. The Crypto Object keeps the state of the AEAD operation until it's finalized or deleted. Once the AEADFinal function is executed successfully, the Crypto Object state returns to the state immediately after the previous AEADInit function.

When P1 equals P1_AEAD_INT_IV and P2 equals P1_ENCRYPT, TLV[TAG_5] must includes the length of the initialization vector. In that case, the initialization vector is generated internally and passed back in the response command. When the device is in FIPS mode (see FIPS Compliance), P1 equal to P1_AEAD will result in SW_CONDITIONS_NOT_SATISFIED.

Command to Applet

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	SE05x_INS_t
P1	P1_AEAD or P1_AEAD_INT_IV	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 AESKey Secure object.
	TLV[TAG_2]	2-byte Crypto Object identifier
	TLV[TAG_5]	Byte array containing the initialization vector (if P1 equals P1_AEAD or P1 equals P1_AEAD and P2 equals P2_DECRYPT) or 2-byte value containing the initialization vector length (if P1 equals P1_AEAD_INT_IV and P2 equals P2_ENCRYPT) [Optional] [Conditional: required when P1 equals P1_AEAD_INT_IV and P2 equals P2_ENCRYPT]
Le	*	

R-APDU Body

Value	Description
TLV[TAG_3]	Byte array containing the used initialization vector. It remains valid until deselect, AEADInit, AEADFinal or AEADOneShot is called. [Conditional: Only when P1 equals P1_AEAD_INT_IV and P2 equals P2_ENCRYPT]

R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

Parameters

- **session_ctx – [in]** The session context
- **objectID – [in]** The object id
- **cipherMode – [in]** The cipher mode
- **cryptoObjectID – [in]** The crypto object id
- **pIV – [in]** { parameter_description }
- **IVLen – [in]** The iv length
- **operation – [in]** The operation

Returns

The sm status.

10.15.4.83 Function Se05x_API_AeadOneShot

- Defined in [File se05x_04_xx_APDU_apis.h](#)

10.15.4.83.1 Function Documentation

smStatus_t Se05x_API_AeadOneShot(*pSe05xSession_t* session_ctx, *uint32_t* objectID, *SE05x_CipherMode_t* cipherMode, *const uint8_t **inputData, *size_t* inputDataLen, *const uint8_t **aad, *size_t* aadLen, *uint8_t **IV, *size_t* IVLen, *uint8_t*

`*tagData, size_t *tagDataLen, uint8_t *outputData, size_t *poutputDataLen, const SE05x_Cipher_Oper_OneShot_t operation)`

Se05x_API_AeadOneShot

Authenticated encryption or decryption with associated data in one shot mode.

The key object must be either an AES key or DES key.

The AEADOneShot command returns the computed GMAC (when P2 equals P2_ENCRYPT_ONESHOT) or indicates whether the GMAC is correct (when P2 equals P2_DECRYPT_ONESHOT). The length of the GMAC is always 16 bytes when P2 equals P2_ENCRYPT_ONESHOT.

When P2 equals P2_DECRYPT_ONESHOT:

- the minimum tag length to pass is 4 bytes.
- when the GMAC tag is not correct, only the result will be returned, no output data will be present.

Note: on applet v4.4.0, the maximum lengths are not yet enforced and might differ from the values listed in the C-APDU.

Command to Applet

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	SE05x_INS_t
P1	P1_AEAD or P1_AEAD_INT_IV	See SE05x_P1_t
P2	P2_ENCRYPT_ONESHOT or P2_DECRYPT_ONESHOT	See SE05x_P2_t
Lc	#(Payload)	
Payload	TLV[TAG_1]	4-byte identifier of the AESKey Secure object.
	TLV[TAG_2]	1-byte AEADMode
	TLV[TAG_3]	Byte array containing input data. Maximum length = 256 bytes. [Optional]
	TLV[TAG_4]	Byte array containing Additional Authenticated Data. Maximum length = 64 bytes. [Optional]
	TLV[TAG_5]	Byte array containing an initialization vector (if P1 equals P1_AEAD) or 2-byte value containing the initialization vector length (if P1 equals P1_AEAD_SP800_108). Maximum IV length = 60 bytes. [Optional] [Conditional: required when P1 equals P1_AEAD_INT_IV]
	TLV[TAG_6]	Byte array containing the GMAC tag to verify. [Conditional: when P2 equals P2_DECRYPT_ONESHOT]
Le	0x00	Expecting return data.

R-APDU Body

Value	Description
TLV[TAG_1]	Byte array containing output data.
TLV[TAG_2]	Byte array containing tag (if P2 = P2_ENCRYPT_ONESHOT) or byte array containing Result (if P2 = P2_DECRYPT_ONESHOT)
TLV[TAG_3]	Byte array containing the initialization vector (if P1 = P1_AEAD_INT_IV and P2 = P2_ENCRYPT_ONESHOT).

R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

Parameters

- **session_ctx** – [in] The session context
- **objectID** – [in] The object id
- **cipherMode** – [in] The cipher mode
- **inputData** – [in] The input data
- **inputDataLen** – [in] The input data length
- **aad** – [in] The aad
- **aadLen** – [in] The aad length
- **IV** – [in] The iv
- **IVLen** – [in] The iv length
- **tagData** – The tag data
- **tagDataLen** – The tag data length (When operation equals P2_ENCRYPT_ONE_SHOT, default tag length is 16.)
- **outputData** – The output data
- **pOutputDataLen** – The poutput data length
- **operation** – [in] The operation

Returns

The sm status.

10.15.4.84 Function Se05x_API_AeadUpdate

- Defined in [File se05x_04_xx_APDU_apis.h](#)

10.15.4.84.1 Function Documentation

smStatus_t Se05x_API_AeadUpdate(*pSe05xSession_t* session_ctx, *SE05x_CryptoObjectID_t* cryptoObjectID, const *uint8_t* **pInputData*, *size_t* *inputDataLen*, *uint8_t* **pOutputData*, *size_t* **pOutputLen*)

Se05x_API_AeadUpdate.

Command to Applet

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	SE05x_INS_t
P1	P1_AEAD	See SE05x_P1_t
P2	P2_UPDATE	See SE05x_P2_t
Lc	#(Payload)	
Payload	TLV[TAG_2]	2-byte Crypto Object identifier
	TLV[TAG_3]	Byte array containing input data [Conditional: only when TLV[TAG_4] is not present] [Optional]
Le	0x00	Expecting returned data.

R-APDU Body

Value	Description
TLV[TAG_1]	Output data [Conditional: only when TLV[TAG_3] is passed as input]

*R-APDU Trailer***Parameters**

- **session_ctx** – **[in]** The session context
- **cryptoObjectID** – **[in]** The crypto object id
- **pInputData** – **[in]** The input data
- **inputDataLen** – **[in]** The input data length
- **pOutputData** – The output data
- **pOutputLen** – The output length

Returns

The sm status.

10.15.4.85 Function Se05x_API_AeadUpdate_aad

- Defined in [File se05x_04_xx_APDU_apis.h](#)

10.15.4.85.1 Function Documentation

smStatus_t Se05x_API_AeadUpdate_aad(*pSe05xSession_t session_ctx, SE05x_CryptoObjectID_t cryptoObjectID, const uint8_t *pAaData, size_t aadDataLen*)

Se05x_API_AeadUpdate_aad

Update a Crypto Object of type CC_AEAD.

The user either needs to send input data or Additional Authenticated Data (AAD), but not both at once.

Command to Applet

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	SE05x_INS_t
P1	P1_AEAD	See SE05x_P1_t
P2	P2_UPDATE	See SE05x_P2_t
Lc	#(Payload)	
Payload	TLV[TAG_2]	2-byte Crypto Object identifier
	TLV[TAG_4]	Byte array containing Additional Authenticated Data. [Conditional: only when TLV[TAG_3] is not present] [Optional]
Le	0x00	Expecting returned data.

R-APDU Body

NA

R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

Parameters

- **session_ctx – [in]** The session context
- **cryptoObjectID – [in]** The crypto object id
- **pAadData – [in]** The aad data
- **aadDataLen – [in]** The aad data length

Returns

The sm status.

10.15.4.86 Function Se05x_API_CheckObjectExists

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.86.1 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 SE05x_INS_t
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_EXIST	See SE05x_P2_t
Lc	#(Payload)	
	TLV[TAG_1]	4-byte existing Secure Object identifier.
Le	0x00	

R-APDU Body

Value	Description
TLV[TAG_1]	1-byte SE05x_Result_t

R-APDU Trailer

SW	Description
SW_NO_ERROR	Data is returned successfully.

Parameters

- **session_ctx – [in]** Session Context [0:kSE05x_pSession]
- **objectID – [in]** object id [1:kSE05x_TAG_1]
- **presult – [out]** [0:kSE05x_TAG_1]

10.15.4.87 Function Se05x_API_CipherFinal

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.87.1 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	SE05x_INS_t
P1	P1_CIPHER	See SE05x_P1_t
P2	P2_FINAL	See SE05x_P2_t
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

- `session_ctx` – [in] Session Context [0:kSE05x_pSession]
- `cryptoObjectID` – [in] cryptoObjectID [1:kSE05x_TAG_2]
- `inputData` – [in] inputData [2:kSE05x_TAG_3]
- `inputDataLen` – [in] Length of inputData
- `outputData` – [out] [0:kSE05x_TAG_1]
- `poutputDataLen` – [inout] Length for outputData

10.15.4.88 Function Se05x_API_CipherInit

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.88.1 Function Documentation

`smStatus_t Se05x_API_CipherInit(pSe05xSession_t session_ctx, uint32_t objectID, SE05x_CryptoObjectID_t cryptoObjectID, 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

- **session_ctx – [in]** Session Context [0:kSE05x_pSession]
- **objectID – [in]** objectID [1:kSE05x_TAG_1]
- **cryptoObjectID – [in]** cryptoObjectID [2:kSE05x_TAG_2]
- **IV – [in]** IV [3:kSE05x_TAG_4]
- **IVLen – [in]** Length of IV
- **operation – [in]** See [SE05x_Cipher_Oper_t](#)

10.15.4.89 Function Se05x_API_CipherOneShot

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.89.1 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, 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)	
Payload	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.

Parameters

- **session_ctx – [in]** The session context
- **objectId – [in]** The object id
- **cipherMode – [in]** The cipher mode
- **inputData – [in]** The input data
- **inputDataLen – [in]** The input data length
- **IV – [in]** Initial vector
- **IVLen – [in]** The iv length
- **outputData –** The output data
- **poutputDataLen –** The poutput data length
- **operation – [in]** The operation

Returns

The sm status.

10.15.4.90 Function Se05x_API_CipherUpdate

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.90.1 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_CRYPTO	SE05x_INS_t
P1	P1_CIPHER	See SE05x_P1_t
P2	P2_UPDATE	See SE05x_P2_t
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

- **session_ctx – [in]** Session Context [0:kSE05x_pSession]
- **cryptoObjectID – [in]** cryptoObjectID [1:kSE05x_TAG_2]
- **inputData – [in]** inputData [2:kSE05x_TAG_3]
- **inputDataLen – [in]** Length of inputData
- **outputData – [out]** [0:kSE05x_TAG_1]
- **poutputDataLen – [inout]** Length for outputData

10.15.4.91 Function Se05x_API_CloseSession

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.91.1 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 SE05x_INS_t
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_SESSION_CLOSE	See SE05x_P2_t

R-APDU Body

NA

R-APDU Trailer

SW	Description
SW_NO_ERROR	The session is closed successfully.

Parameters

`session_ctx` – [in] Session Context [0:kSE05x_pSession]

10.15.4.92 Function Se05x_API_CreateCounter

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.92.1 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 SE05x_P1_t
P2	P2_DEFAULT	See SE05x_P2_t
Payload	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 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

- **session_ctx** – [in] Session Context [0:kSE05x_pSession]
- **policy** – [in] policy [1:kSE05x_TAG_POLICY]
- **objectID** – [in] object id [2:kSE05x_TAG_1]
- **size** – [in] size [3:kSE05x_TAG_2]

10.15.4.93 Function Se05x_API_CreateCryptoObject

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.93.1 Function Documentation

`smStatus_t Se05x_API_CreateCryptoObject(pSe05xSession_t session_ctx, SE05x_CryptoObjectID_t cryptoObjectID, 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 SE05x_INS_t
P1	P1_CRYPTO_OBJ	See SE05x_P1_t
P2	P2_DEFAULT	See SE05x_P2_t
Lc	#(Payload)	Payload length
Payload	TLV[TAG_1]	2-byte Crypto Object identifier
	TLV[TAG_2]	1-byte SE05x_CryptoObject_t
	TLV[TAG_3]	1-byte Crypto Object subtype, either from DigestModeRef, Cipher Mode, 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

- **session_ctx** – [in] Session Context [0:kSE05x_pSession]
- **cryptoObjectID** – [in] cryptoObjectID [1:kSE05x_TAG_1]
- **cryptoContext** – [in] cryptoContext [2:kSE05x_TAG_2]
- **subtype** – [in] 1-byte Crypto Object subtype, either from DigestMode, Cipher Mode or MACAlgo (depending on TAG_2). [3:kSE05x_TAG_3]

10.15.4.94 Function Se05x_API_CreateECCurve

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.94.1 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

- **session_ctx** – [in] Session Context [0:kSE05x_pSession]
- **curveID** – [in] curve id [1:kSE05x_TAG_1]

10.15.4.95 Function Se05x_API_CreateSession

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.95.1 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 SE05x_INS_t
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_SESSION_CREATE	See SE05x_P2_t
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

- **session_ctx** – [in] Session Context [0:kSE05x_pSession]
- **authObjectID** – [in] auth [1:kSE05x_TAG_1]
- **sessionId** – [out] [0:kSE05x_TAG_1]
- **psessionIdLen** – [inout] Length for sessionId

10.15.4.96 Function Se05x_API_DeleteAll

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.96.1 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 RESERVED_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 SE05x_INS_t
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_DELETE_ALL	See SE05x_P2_t
Lc	0x00	

R-APDU Body

NA

R-APDU Trailer

SW	Description
SW_NO_ERROR	Data is returned successfully.

Parameters

`session_ctx – [in] Session Context [0:kSE05x_pSession]`

10.15.4.97 Function Se05x_API_DeleteAll_Iterative

- Defined in [File se05x_APDU.h](#)

10.15.4.97.1 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.

Parameters

`session_ctx – [in] Session Context`

Returns

The status of API.

10.15.4.98 Function Se05x_API_DeleteCryptoObject

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.98.1 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 SE05x_INS_t
P1	P1_CRYPTO_OBJ	See SE05x_P1_t
P2	P2_DELETE_OBJECT	See SE05x_P2_t
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

- session_ctx – [in]** Session Context [0:kSE05x_pSession]
- cryptoObjectID – [in]** cryptoObjectID [1:kSE05x_TAG_1]

10.15.4.99 Function Se05x_API_DeleteECCurve

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.99.1 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 SE05x_INS_t
P1	P1_CURVE	See SE05x_P1_t

Field	Value	Description
P2	P2_DELETE_OBJECT	See SE05x_P2_t
Lc	#(Payload)	
	TLV[TAG_1]	1-byte curve identifier (from SE05x_ECCurve_t)

R-APDU Body

NA

R-APDU Trailer

SW	Description
SW_NO_ERROR	Data is returned successfully.

Parameters

- **session_ctx** – [in] Session Context [0:kSE05x_pSession]
- **curveID** – [in] curve id [1:kSE05x_TAG_1]

10.15.4.100 Function Se05x_API_DeleteSecureObject

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.100.1 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 SE05x_INS_t
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_DELETE_OBJECT	See SE05x_P2_t
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

- **session_ctx** – [in] Session Context [0:kSE05x_pSession]
- **objectID** – [in] object id [1:kSE05x_TAG_1]

10.15.4.101 Function Se05x_API_DFAuthenticateFirstPart1

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.101.1 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_CRYPTO	SE05x_INS_t
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_AUTH_FIRST_PART1	See SE05x_P2_t
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

- **session_ctx – [in]** Session Context [0:kSE05x_pSession]
- **objectID – [in]** objectID [1:kSE05x_TAG_1]
- **inputData – [in]** inputData [2:kSE05x_TAG_2]
- **inputDataLen – [in]** Length of inputData
- **outputData – [out]** [0:kSE05x_TAG_1]
- **poutputDataLen – [inout]** Length for outputData

10.15.4.102 Function Se05x_API_DFAuthenticateFirstPart2

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.102.1 Function Documentation

smStatus_t Se05x_API_DFAuthenticateFirstPart2(*pSe05xSession_t session_ctx, const uint8_t *inputData, size_t input DataLen, 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. DFAuthenticateFirst Part1 needs to be called before; otherwise an error is returned.

Command to Applet

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	SE05x_INS_t
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_AUTH_FIRST_PART2	See SE05x_P2_t
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

- **session_ctx** – **[in]** Session Context [0:kSE05x_pSession]
- **inputData** – **[in]** inputData [1:kSE05x_TAG_1]
- **inputDataLen** – **[in]** Length of inputData
- **outputData** – **[out]** [0:kSE05x_TAG_1]
- **poutputDataLen** – **[inout]** Length for outputData

10.15.4.103 Function Se05x_API_DFAuthenticateNonFirstPart1

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.103.1 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_CRYPTO	SE05x_INS_t
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_AUTH_NONFIRST_PART1	See SE05x_P2_t
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

- **session_ctx** – **[in]** Session Context [0:kSE05x_pSession]
- **objectID** – **[in]** objectID [1:kSE05x_TAG_1]
- **inputData** – **[in]** inputData [2:kSE05x_TAG_2]
- **inputDataLen** – **[in]** Length of inputData
- **outputData** – **[out]** [0:kSE05x_TAG_1]
- **poutputDataLen** – **[inout]** Length for outputData

10.15.4.104 Function Se05x_API_DFAuthenticateNonFirstPart2

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.104.1 Function Documentation

smStatus_t **Se05x_API_DFAuthenticateNonFirstPart2**(*pSe05xSession_t session_ctx, const uint8_t *inputData, size_t inputDataLen*)

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_CRYPTO	SE05x_INS_t
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_AUTH_NONFIRST_PART2	See SE05x_P2_t
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.

Parameters

- session_ctx** – [in] Session Context [0:kSE05x_pSession]
- inputData** – [in] inputData [1:kSE05x_TAG_1]
- inputDataLen** – [in] Length of inputData

10.15.4.105 Function Se05x_API_DFCChangeKeyPart1

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.105.1 Function Documentation

smStatus_t **Se05x_API_DFCChangeKeyPart1**(*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_DFCChangeKeyPart1

The DFCChangeKeyPart1 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_CRYPTO	SE05x_INS_t
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_CHANGE_KEY_PART1	See SE05x_P2_t
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

- **session_ctx** – **[in]** Session Context [0:kSE05x_pSession]
- **oldObjectID** – **[in]** oldObjectID [1:kSE05x_TAG_1]
- **newObjectID** – **[in]** newObjectID [2:kSE05x_TAG_2]
- **keySetNr** – **[in]** keySetNr [3:kSE05x_TAG_3]
- **keyNoDESFire** – **[in]** keyNoDESFire [4:kSE05x_TAG_4]
- **keyVer** – **[in]** keyVer [5:kSE05x_TAG_5]
- **KeyData** – **[out]** [0:kSE05x_TAG_1]
- **pKeyDataLen** – **[inout]** Length for KeyData

10.15.4.106 Function Se05x_API_DFChangeKeyPart2

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.106.1 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_CRYPTO	SE05x_INS_t
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_CHANGE_KEY_PART2	See SE05x_P2_t
Lc	#(Payload)	
	TLV[TAG_1]	MAC
Le	0x00	

R-APDU Body

Value	Description
TLV[TAG_1]	1-byte SE05x_Result_t

R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

Parameters

- **session_ctx** – **[in]** Session Context [0:kSE05x_pSession]
- **MAC** – **[in]** MAC [1:kSE05x_TAG_1]
- **MACLen** – **[in]** Length of MAC
- **presult** – **[out]** [0:kSE05x_TAG_1]

10.15.4.107 Function Se05x_API_DFDiversifyKey

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.107.1 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_CRYPTO	SE05x_INS_t
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_DIVERSIFY	See SE05x_P2_t
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

- **session_ctx** – [in] Session Context [0:kSE05x_pSession]
- **masterKeyID** – [in] masterKeyID [1:kSE05x_TAG_1]
- **diversifiedKeyID** – [in] diversifiedKeyID [2:kSE05x_TAG_2]
- **divInputData** – [in] divInputData [3:kSE05x_TAG_3]
- **divInputDataLen** – [in] Length of divInputData

10.15.4.108 Function Se05x_API_DFDumpSessionKeys

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.108.1 Function Documentation

smStatus_t Se05x_API_DFDumpSessionKeys([pSe05xSession_t](#) session_ctx, [uint8_t](#) *sessionData, [size_t](#) *psessionDataLen)

Se05x_API_DFDumpSessionKeys

Dump the Transaction Identifier and the session keys to the host.

Command to Applet

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	SE05x_INS_t
P1	P1_DEFAULT	See SE05x_P1_t

Field	Value	Description
P2	P2_DUMP_KEY	See SE05x_P2_t
Lc	#(Payload)	
Le	0x28	Expecting TLV with 38 bytes data.

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

- **session_ctx** – [in] Session Context [0:kSE05x_pSession]
- **sessionData** – [out] 38 bytes: KeyID.SesAuthENCKey || KeyID.SesAuthMACKey || TI || Cmd-Ctr [0:kSE05x_TAG_1]
- **pSessionDataLen** – [inout] Length for sessionData

10.15.4.109 Function Se05x_API_DFKillAuthentication

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.109.1 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_CRYPTO	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

session_ctx – [in] Session Context [0:kSE05x_pSession]

10.15.4.110 Function Se05x_API_DigestFinal

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.110.1 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 SE05x_INS_t
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_FINAL	See SE05x_P2_t
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

- session_ctx – [in]** Session Context [0:kSE05x_pSession]
- cryptoObjectID – [in]** cryptoObjectID [1:kSE05x_TAG_2]
- inputData – [in]** inputData [2:kSE05x_TAG_3]
- inputDataLen – [in]** Length of inputData
- cmacValue – [out]** [0:kSE05x_TAG_1]
- pcmacValueLen – [inout]** Length for cmacValue

10.15.4.111 Function Se05x_API_DigestInit

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.111.1 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_CRYPTO	See SE05x_INS_t
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_INIT	See SE05x_P2_t
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

- **session_ctx** – [in] Session Context [0:kSE05x_pSession]
- **cryptoObjectID** – [in] cryptoObjectID [1:kSE05x_TAG_2]

10.15.4.112 Function Se05x_API_DigestOneShot

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.112.1 Function Documentation

[**smStatus_t Se05x_API_DigestOneShot\(pSe05xSession_t session_ctx, uint8_t digestMode, const uint8_t *inputData, size_t inputDataLen, uint8_t *hashValue, size_t *phashValueLen\)**](#)

Se05x_API_DigestOneShot

Performs a hash operation in one shot (without context).

Command to Applet

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	See SE05x_INS_t
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_ONESHOT	See SE05x_P2_t
Lc	#(Payload)	
	TLV[TAG_1]	1-byte DigestMode (except DIGEST_NO_HASH)

Field	Value	Description
	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

- **session_ctx** – [in] Session Context [0:kSE05x_pSession]
- **digestMode** – [in] digestMode [1:kSE05x_TAG_1]
- **inputData** – [in] inputData [2:kSE05x_TAG_2]
- **inputDataLen** – [in] Length of inputData
- **hashValue** – [out] [0:kSE05x_TAG_1]
- **phashValueLen** – [inout] Length for hashValue

10.15.4.113 Function Se05x_API_DigestUpdate

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.113.1 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 SE05x_INS_t
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_UPDATE	See SE05x_P2_t
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

- **session_ctx** – [in] Session Context [0:kSE05x_pSession]
- **cryptoObjectID** – [in] cryptoObjectID [1:kSE05x_TAG_2]
- **inputData** – [in] inputData [2:kSE05x_TAG_3]
- **inputDataLen** – [in] Length of inputData

10.15.4.114 Function Se05x_API_DisableObjCreation

- Defined in [File se05x_04_xx_APDU_apis.h](#)

10.15.4.114.1 Function Documentation

smStatus_t **Se05x_API_DisableObjCreation**(*pSe05xSession_t session_ctx*, *SE05x_LockIndicator_t lockIndicator*, *SE05x_RestrictMode_t restrictMode*)

Se05x_API_DisableObjCreation

Command to Applet

R-APDU Body

NA

R-APDU Trailer

Parameters

- **session_ctx** – [in] Session Context [0:kSE05x_pSession]
- **lockIndicator** – [in] [1:kSE05x_TAG_1]
- **restrictMode** – [in] [2:kSE05x_TAG_2]

10.15.4.115 Function Se05x_API_EC_CurveGetId

- Defined in [File se05x_APDU.h](#)

10.15.4.115.1 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_GetECCurvid](#) but uses [SE05x_ECCurve_t](#) as a type instead of [uint8_t](#).

Parameters

- **session_ctx** – [in] The session context
- **objectID** – [in] The object id
- **pcurveId** – The pcurve identifier

Returns

The sm status.

10.15.4.116 Function Se05x_API_ECDHGenerateSharedSecret

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.116.1 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_CRYPTO	SE05x_INS_t
P1	P1_EC	See SE05x_P1_t
P2	P2_DH	See SE05x_P2_t
Lc	#(Payload)	
Payload	TLV[TAG_1]	4-byte identifier of the key pair or private key.
TLV[TAG_2]	External public key (see <code>ECKeyRef</code>).	
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

- **session_ctx** – **[in]** Session Context [0:kSE05x_pSession]
- **objectID** – **[in]** objectID [1:kSE05x_TAG_1]
- **pubKey** – **[in]** pubKey [2:kSE05x_TAG_2]
- **pubKeyLen** – **[in]** Length of pubKey
- **sharedSecret** – **[out]** [0:kSE05x_TAG_1]
- **psharedSecretLen** – **[inout]** Length for sharedSecret

10.15.4.117 Function Se05x_API_ECDHGenerateSharedSecret_InObject

- Defined in [File se05x_04_xx_APDU_apis.h](#)

10.15.4.117.1 Function Documentation

smStatus_t Se05x_API_ECDHGenerateSharedSecret_InObject(*pSe05xSession_t session_ctx, uint32_t objectID, const uint8_t *pubKey, size_t pubKeyLen, uint32_t sharedSecretID, uint8_t invertEndianness*)

Se05x_API_ECDHGenerateSharedSecret_InObject

See [Se05x_API_ECDHGenerateSharedSecret](#)

10.15.4.118 Function Se05x_API_ECDHGenerateSharedSecret_InObject_extended

- Defined in [File se05x_04_xx_APDU_apis.h](#)

10.15.4.118.1 Function Documentation

smStatus_t Se05x_API_ECDHGenerateSharedSecret_InObject_extended(*pSe05xSession_t session_ctx, uint32_t objectID, const uint8_t *pubKey, size_t pubKeyLen, SE05x_ECDHAlgo_t ecdhAlgo, uint32_t sharedSecretID, uint8_t invertEndianness*)

Se05x_API_ECDHGenerateSharedSecret_InObject_extended

See [Se05x_API_ECDHGenerateSharedSecret_InObject_extended](#). New ECDH api with support for ECDH algo input (EC_SVDP_DH and EC_SVDP_DH_PLAIN).

Parameters

- **session_ctx** – **[in]** The session context
- **objectID** – **[in]** Private key or key pair identifier
- **pubKey** – **[in]** External EC public key
- **pubKeyLen** – **[in]** External EC public key length
- **ecdhAlgo** – **[in]** ECDH Algorithm
- **sharedSecretID** – **[in]** Identifier to store derived key
- **invertEndianness** – **[in]** Option to invert endianness of derived key

10.15.4.119 Function Se05x_API_ECDSASign

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.119.1 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	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 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

- **session_ctx** – **[in]** Session Context [0:kSE05x_pSession]
 - **objectID** – **[in]** objectID [1:kSE05x_TAG_1]
 - **ecSignAlgo** – **[in]** ecSignAlgo [2:kSE05x_TAG_2]
 - **inputData** – **[in]** inputData [3:kSE05x_TAG_3]
 - **inputDataLen** – **[in]** Length of inputData
 - **signature** – **[out]** [0:kSE05x_TAG_1]
 - **psignatureLen** – **[inout]** Length for signature

10.15.4.120 Function Se05x_API_ECDSAVerify

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.120.1 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

- **session_ctx** – [in] Session Context [0:kSE05x_pSession]
- **objectID** – [in] objectID [1:kSE05x_TAG_1]
- **ecSignAlgo** – [in] ecSignAlgo [2:kSE05x_TAG_2]
- **inputData** – [in] inputData [3:kSE05x_TAG_3]
- **inputDataLen** – [in] Length of inputData
- **signature** – [in] signature [4:kSE05x_TAG_5]
- **signatureLen** – [in] Length of signature
- **presult** – [out] [0:kSE05x_TAG_1]

10.15.4.121 Function Se05x_API_ECPPointMultiply_InputObj

- Defined in [File se05x_xx_APDU_apis.h](#)

10.15.4.121.1 Function Documentation

smStatus_t Se05x_API_ECPPointMultiply_InputObj(*pSe05xSession_t session_ctx, uint32_t objectID, uint32_t pubKeyID, uint32_t sharedSecretID, uint8_t *sharedSecretOuput, size_t *psharedSecretOuputLen, SE05x_ECPMAlgo_t ECPMAlgo*)

Se05x_API_ECPPointMultiply_InputObj.

Performs an EC Point Multiply operation using the specified input objects

Parameters

- **session_ctx** – [in] The session context
- **objectID** – [in] Private key or key pair identifier
- **pubKeyID** – [in] public key identifier
- **sharedSecretID** – [in] Identifier to store derived key
- **sharedSecretOuput** – [in] Output Buffer to store the result
- **psharedSecretOuputLen** – [in] Length of sharedSecret Output BUffer
- **ECPMAlgo** – [in] Algorithm used for the EC Point Operation

10.15.4.122 Function Se05x_API_EdDSASign

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.122.1 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_CRYPTO	SE05x_INS_t

Field	Value	Description
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 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

- **session_ctx** – [in] Session Context [0:kSE05x_pSession]
- **objectId** – [in] objectID [1:kSE05x_TAG_1]
- **edSignAlgo** – [in] edSignAlgo [2:kSE05x_TAG_2]
- **inputData** – [in] inputData [3:kSE05x_TAG_3]
- **inputDataLen** – [in] Length of inputData
- **signature** – [out] [0:kSE05x_TAG_1]
- **psignatureLen** – [inout] Length for signature

10.15.4.123 Function Se05x_API_EdDSAVerify

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.123.1 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 EDSSignatureAlgoRef.
	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 (SE05x_Result_t).

R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.
SW_CONDITIONS_NOT_SATISFIED	Incorrect data

Parameters

- **session_ctx** – [in] Session Context [0:kSE05x_pSession]
- **objectID** – [in] objectID [1:kSE05x_TAG_1]
- **edSignAlgo** – [in] edSignAlgo [2:kSE05x_TAG_2]
- **inputData** – [in] inputData [3:kSE05x_TAG_3]
- **inputDataLen** – [in] Length of inputData
- **signature** – [in] signature [4:kSE05x_TAG_5]
- **signatureLen** – [in] Length of signature
- **presult** – [out] [0:kSE05x_TAG_1]

10.15.4.124 Function Se05x_API_ExchangeSessionData

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.124.1 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 SE05x_INS_t
P1	P1_DEFAULT	See SE05x_P1_t
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

- **session_ctx** – [in] Session Context [0:kSE05x_pSession]
- **policy** – [in] Check pdf [1:kSE05x_TAG_1]

10.15.4.125 Function Se05x_API_ExportObject

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.125.1 Function Documentation

[**smStatus_t Se05x_API_ExportObject\(pSe05xSession_t session_ctx, uint32_t objectID, SE05x_RSAKeyComponent_t rsaKeyComp, uint8_t *data, size_t *pdataLen\)**](#)

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 SE05x_INS_t .
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_EXPORT	See SE05x_P2_t
Lc	#(Payload)	Payload Length.
	TLV[TAG_1]	4-byte object identifier
	TLV[TAG_2]	1-byte SE05x_RSAKeyComponent_t (only applies to Secure Objects of type RSAKey).
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

- **session_ctx – [in]** Session Context [0:kSE05x_pSession]
- **objectID – [in]** object id [1:kSE05x_TAG_1]
- **rsaKeyComp – [in]** rsaKeyComp [2:kSE05x_TAG_2]
- **data – [out]** [0:kSE05x_TAG_1]
- **pdataLen – [inout]** Length for data

10.15.4.126 Function Se05x_API_GetECCurveld

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.126.1 Function Documentation

smStatus_t Se05x_API_GetECCurveld([pSe05xSession_t](#) session_ctx, uint32_t objectID, uint8_t *pcurveld)

Se05x_API_GetECCurveld

Get the curve associated with an EC key.

Command to Applet

Field	Value	Description
CLA	0x80	
INS	INS_READ	See SE05x_INS_t
P1	P1_CURVE	See SE05x_P1_t
P2	P2_ID	See SE05x_P2_t
Lc	#(Payload)	
Payload	TLV[TAG_1]	4-byte identifier
Le	0x00	

R-APDU Body

Value	Description
TLV[TAG_1]	1-byte curve identifier (from SE05x_ECCurve_t)

R-APDU Trailer

SW	Description
SW_NO_ERROR	Data is returned successfully.

Parameters

- **session_ctx** – [in] Session Context [0:kSE05x_pSession]
- **objectID** – [in] object id [1:kSE05x_TAG_1]
- **pcurveId** – [out] [0:kSE05x_TAG_1]

10.15.4.127 Function Se05x_API_GetExtVersion

- Defined in [File se05x_04_xx_APDU_apis.h](#)

10.15.4.127.1 Function Documentation

[**smStatus_t Se05x_API_GetExtVersion\(pSe05xSession_t session_ctx, uint8_t *appletVersion, size_t *appletVersionLen\)**](#)

Se05x_API_GetExtVersion

Gets the applet extended version information.

This will return 37-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 SE05x_INS_t
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_VERSION or P2_VERSION_EXT	See SE05x_P2_t
Lc	#(Payload)	

Field	Value	Description
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_1]	7-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.

Parameters

- **session_ctx – [in]** The session context
- **pappletVersion –** The applet version
- **appletVersionLen –** The applet version length

Returns

The sm status.

10.15.4.128 Function Se05x_API_GetFreeMemory

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.128.1 Function Documentation

`smStatus_t Se05x_API_GetFreeMemory(pSe05xSession_t session_ctx, SE05x_MemoryType_t memoryType, uint32_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 SE05x_INS_t
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_MEMORY	See SE05x_P2_t
Lc	#(Payload)	
	TLV[TAG_1]	SE05x_MemTyp_t
Le	0x04	Expecting TLV with 2-byte data.

R-APDU Body

Value	Description
TLV[TAG_1]	2 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.

Parameters

- `session_ctx` – [in] The session context
- `memoryType` – [in] The memory type
- `pfreeMem` – The pfree memory

Returns

The sm status.

10.15.4.129 Function Se05x_API_GetRandom

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.129.1 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 SE05x_INS_t
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_RANDOM	See SE05x_P2_t
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.

Parameters

- `session_ctx` – [in] The session context
- `size` – [in] The size
- `randomData` – The random data
- `prandomDataLen` – The prandom data length

Returns

The sm status.

10.15.4.130 Function Se05x_API_GetTimestamp

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.130.1 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 SE05x_INS_t
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_TIME	See SE05x_P2_t
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.

Parameters

- `session_ctx` – [in] The session context
- `ptimeStamp` – The ptime stamp

Returns

The sm status.

10.15.4.131 Function Se05x_API_GetVersion

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.131.1 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 SE05x_INS_t
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_VERSION or P2_VERSION_EXT	See SE05x_P2_t
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_1]	7-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.

Parameters

- session_ctx – [in]** The session context
- pappletVersion –** The applet version
- appletVersionLen –** The applet version length

Returns

The sm status.

10.15.4.132 Function Se05x_API_HKDF

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.132.1 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 *hkdfOuput, size_t *phkdfOuputLen)`

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_CRYPTO	SE05x_INS_t
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_HKDF	See SE05x_P2_t
Lc	#(Payload)	
Payload	TLV[TAG_1]	4-byte HMACKey identifier (= IKM)
TLV[TAG_2]	1-byte DigestMode (except DIGEST_NO_HASH)	
TLV[TAG_3]	Byte array (0-64 bytes) containing salt. [Optional] [Conditional: only when TLV[TAG_6] is absent.]	
TLV[TAG_4]	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]	4-byte HMACKey identifier containing salt. [Optional] [Conditional: only when TLV[TAG_3] is absent]	
TLV[TAG_7]	4-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

- **session_ctx** – [in] Session Context [0:kSE05x_pSession]
- **hmacID** – [in] hmacID [1:kSE05x_TAG_1]
- **digestMode** – [in] digestMode [2:kSE05x_TAG_2]
- **salt** – [in] salt [3:kSE05x_TAG_3]
- **saltLen** – [in] Length of salt
- **info** – [in] info [4:kSE05x_TAG_4]
- **infoLen** – [in] Length of info
- **deriveDataLen** – [in] 2-byte requested length (L) [5:kSE05x_TAG_5]
- **hkdfOuput** – [out] [0:kSE05x_TAG_1]
- **phkdfOuputLen** – [inout] Length for hkdfOuput

10.15.4.133 Function Se05x_API_HKDF_Extended

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.133.1 Function Documentation

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_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_CRYPTO	SE05x_INS_t
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_HKDF_EXPAND_ONLY	See SE05x_P2_t
Lc	#(Payload)	
Payload	TLV[TAG_1]	4-byte HMACKey identifier (= PRK)
TLV[TAG_2]	1-byte DigestMode (except DIGEST_NO_HASH)	
TLV[TAG_3]	Byte array (0-64 bytes) containing IV. [Optional] [Conditional: only when TLV[TAG_6] is absent.]	
TLV[TAG_4]	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	

Field	Value	Description
TLV[TAG_6]	4-byte HMACKey identifier containing IV. [Optional] [Conditional: only when TLV[TAG_3] is absent]	
TLV[TAG_7]	4-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

- **session_ctx – [in]** Session Context [0:kSE05x_pSession]
- **hmacID – [in]** hmacID [1:kSE05x_TAG_1]
- **digestMode – [in]** digestMode [2:kSE05x_TAG_2]
- **hkdfMode – [in]** hkdfMode
- **salt – [in]** salt [3:kSE05x_TAG_3]
- **saltLen – [in]** Length of salt
- **saltID – [in]** saltID
- **info – [in]** info [4:kSE05x_TAG_4]
- **infoLen – [in]** Length of info
- **derivedKeyID – [in]** deriveKeyID
- **deriveDataLen – [in]** 2-byte requested length (L) [5:kSE05x_TAG_5]
- **hkdfOuput – [out]** [0:kSE05x_TAG_1]
- **phkdfOuputLen – [inout]** Length for hkdfOuput

10.15.4.134 Function Se05x_API_I2CM_ExecuteCommandSet

- Defined in [File se05x_APDU/apis.h](#)

10.15.4.134.1 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 *responseLen, SE05x_Time
Stamp_t *ptimeStamp, uint8_t *chipId, size_t *pchipIdLen, uint8_t *signature, size_t *psignatureLen, uint8_t *randomAttst,
size_t randomAttstLen, uint8_t *pObjectSize, size_t *pObjectSizeLen, uint8_t *pCmd, size_t *pCmdLen)
```

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_CRYPTO	See SE05x_INS_t , in addition to INS_CRYPTO, users can set the INS_ATTEST flag. In that case, attestation applies.
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_I2CM	See SE05x_P2_t
Lc	#(Payload)	
	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 SE05x_AttestationAlgo_t [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[TAG_1]	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[TAG_2]	TLV containing 18-byte chip unique ID
TLV[TAG_4]	TLV containing 2-byte total size of the Secure Object
TLV[TAG_TS]	TLV containing 12-byte timestamp
TLV[TAG_ATT_SIG]	TLV 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.

Parameters

- **session_ctx** – [in] The session context
- **inputData** – [in] The input data
- **inputDataLen** – [in] The input data length
- **attestationID** – [in] The attestation id
- **attestationAlgo** – [in] The attestation algorithm
- **response** – The response
- **presponseLen** – The presponse length
- **pTimeStamp** – The ptime stamp
- **chipId** – The chip identifier
- **pchipIdLen** – The pchip identifier length
- **signature** – The signature
- **psignatureLen** – The psignature length
- **randomAttst** – The random attst
- **randomAttstLen** – [in] The random attst length
- **pObjectSize** – The pObjectSize
- **pObjectSizeLen** – The pObjectSize length
- **pCmd** – The pCmd
- **pCmdLen** – The pCmd length

Returns

The sm status.

10.15.4.135 Function Se05x_API_I2CM_Send

- Defined in [File se05x_tlv.h](#)

10.15.4.135.1 Function Documentation

```
smStatus_t Se05x_API_I2CM_Send(pSe05xSession_t sessionId, const uint8_t *buffer, size_t bufferLen, uint8_t *result, size_t *presultLen)
```

10.15.4.136 Function Se05x_API_ImportExternalObject

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.136.1 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 (Write ECKey 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 ECKey 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. A000000396545300000001030000000)
 - 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 ECKeySessionInternalAuthenticate. 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-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, 00000000000000000000000000000000400008001)
- CMACsession key = CMAC(MK, 00000000000000000000000000000000600008001)
- RMACsession key = CMAC(MK, 00000000000000000000000000000000700008001)

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 0x0001) 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,

Receipt = CMAC(RMAC session key, <input from TLV 0xA6 and TLV 0x7F49>)

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	

Field	Value	Description
INS	INS_IMPORT_EXTERNAL	See SE05x_INS_t
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_DEFAULT	See SE05x_P2_t
Lc	#(Payload)	
Payload	TLV[TAG_IMPORT_AUTH_DATA]	Authentication data
	TLV[TAG_IMPORT_AUTH_KEY_ID]	Host 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

- **session_ctx** – **[in]** Session Context [0:kSE05x_pSession]
- **ECKeydata** – **[in]** ECKeydata [1:kSE05x_TAG_2]
- **ECKeydataLen** – **[in]** Length of ECKeydata
- **ECAuthKeyID** – **[in]** ECAuthKeyID
- **ECAuthKeyIDLen** – **[in]** Length of ECAuthKeyID
- **serializedObject** – **[in]** serializedObject [2:kSE05x_TAG_3]
- **serializedObjectLen** – **[in]** Length of serializedObject

10.15.4.137 Function Se05x_API_ImportObject

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.137.1 Function Documentation

smStatus_t Se05x_API_ImportObject(*pSe05xSession_t session_ctx, uint32_t objectID, SE05x_RSAKeyComponent_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 SE05x_P1_t
P2	P2_IMPORT	See SE05x_P2_t
Payload	TLV[TAG_1]	4-byte identifier.
	TLV[TAG_2]	1-byte SE05x_RSAKeyComponent_t [Conditional: only when the identifier refers to an RSAKey object]
	TLV[TAG_3]	Serialized object (encrypted).

R-APDU Body

NA

R-APDU Trailer

Parameters

- **session_ctx** – [in] Session Context [0:kSE05x_pSession]
- **objectID** – [in] object id [1:kSE05x_TAG_1]
- **rsaKeyComp** – [in] rsaKeyComp [2:kSE05x_TAG_2]
- **serializedObject** – [in] serializedObject [3:kSE05x_TAG_3]
- **serializedObjectLen** – [in] Length of serializedObject

10.15.4.138 Function Se05x_API_IncCounter

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.138.1 Function Documentation

smStatus_t Se05x_API_IncCounter(*pSe05xSession_t session_ctx, uint32_t objectID*)

Se05x_API_IncCounter

See [Se05x_API_CreateCounter](#)

Parameters

- **session_ctx** – [in] Session Context [0:kSE05x_pSession]
- **objectID** – [in] object id [1:kSE05x_TAG_1]

10.15.4.139 Function Se05x_API_MACFinal

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.139.1 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	SE05x_INS_t
P1	P1_MAC	See SE05x_P1_t
P2	P2_FINAL	See SE05x_P2_t
Payload	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 SE05x_Result_t (when MACInit had P2 = P2_VERIFY).

R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

Parameters

- **session_ctx** – **[in]** Session Context [0:kSE05x_pSession]
- **inputData** – **[in]** inputData [1:kSE05x_TAG_1]
- **inputDataLen** – **[in]** Length of inputData
- **cryptoObjectID** – **[in]** cryptoObjectID [2:kSE05x_TAG_2]
- **macValidateData** – **[in]** macValidateData [3:kSE05x_TAG_3]
- **macValidateDataLen** – **[in]** Length of macValidateData
- **macValue** – **[out]** [0:kSE05x_TAG_1]
- **pmacValueLen** – **[inout]** Length for macValue

10.15.4.140 Function Se05x_API_MACInit

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.140.1 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

- **session_ctx – [in]** Session Context [0:kSE05x_pSession]
- **objectID – [in]** objectID [1:kSE05x_TAG_1]
- **cryptoObjectID – [in]** cryptoObjectID [2:kSE05x_TAG_2]
- **mac_oper – [in]** The Operation

10.15.4.141 Function Se05x_API_MACOneShot_G

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.141.1 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 Verfication.

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_CRYPTO	SE05x_INS_t
P1	P1_MAC	See SE05x_P1_t
P2	P2_GENERATE_ONESHOT or P2_VALIDATE_ONESHOT	See SE05x_P2_t
Lc	#(Payload)	
Payload	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 SE05x_Result_t (when p2=P2_VALIDATE_ONESHOT).

R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

Parameters

- **session_ctx** – **[in]** Session Context [0:kSE05x_pSession]
- **objectID** – **[in]** objectID [1:kSE05x_TAG_1]
- **macOperation** – **[in]** macOperation [2:kSE05x_TAG_2]
- **inputData** – **[in]** inputData [3:kSE05x_TAG_3]
- **inputDataLen** – **[in]** Length of inputData
- **macValue** – **[out]** [0:kSE05x_TAG_1]
- **pmacValueLen** – **[inout]** Length for macValue

10.15.4.142 Function Se05x_API_MACOneShot_V

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.142.1 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 *result, size_t *presultLen*)

Se05x_API_MACOneShot_V

Validate. See [Se05x_API_MACOneShot_G](#) for Generation.

Parameters

- **session_ctx** – **[in]** Session Context [0:kSE05x_pSession]
- **objectID** – **[in]** objectID [1:kSE05x_TAG_1]
- **macOperation** – **[in]** macOperation [2:kSE05x_TAG_2]
- **inputData** – **[in]** inputData [3:kSE05x_TAG_3]
- **inputDataLen** – **[in]** Length of inputData
- **MAC** – **[in]** MAC to verify (when P2=P2_VALIDATE_ONESHOT) [4:kSE05x_TAG_5]
- **MACLen** – **[in]** Length of MAC
- **result** – **[out]** [0:kSE05x_TAG_1]
- **presultLen** – **[inout]** Length for result

10.15.4.143 Function Se05x_API_MACUpdate

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.143.1 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_CRYPTO	SE05x_INS_t
P1	P1_MAC	See SE05x_P1_t

Field	Value	Description
P2	P2_UPDATE	See SE05x_P2_t
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

- **session_ctx** – [in] Session Context [0:kSE05x_pSession]
- **inputData** – [in] inputData [1:kSE05x_TAG_1]
- **inputDataLen** – [in] Length of inputData
- **cryptoObjectID** – [in] cryptoObjectID [2:kSE05x_TAG_2]

10.15.4.144 Function Se05x_API_PBKDF2

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.144.1 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_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

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	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.

Field	Value	Description
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

- **session_ctx** – [in] Session Context [0:kSE05x_pSession]
- **objectID** – [in] 4-byte password identifier (object type must be HMACKey) [1:kSE05x_TAG_1]
- **salt** – [in] salt [2:kSE05x_TAG_2]
- **saltLen** – [in] Length of salt
- **count** – [in] count [3:kSE05x_TAG_3]
- **requestedLen** – [in] requestedLen [4:kSE05x_TAG_4]
- **derivedSessionKey** – [out] [0:kSE05x_TAG_1]
- **pderivedSessionKeyLen** – [inout] Length for derivedSessionKey

10.15.4.145 Function Se05x_API_PBKDF2_extended

- Defined in [File se05x_04_xx_APDU_apis.h](#)

10.15.4.145.1 Function Documentation

smStatus_t Se05x_API_PBKDF2_extended(*pSe05xSession_t session_ctx, uint32_t objectID, const uint8_t *salt, size_t saltLen, uint32_t saltID, uint16_t count, SE05x_MACAlgo_t macAlgo, uint16_t requestedLen, uint32_t derivedSessionKey ID, uint8_t *derivedSessionKey, size_t *pderivedSessionKeyLen*)

Se05x_API_PBKDF2_extended

See [Se05x_API_PBKDF2_extended](#). New PBKDF2 api with optional salt object id and optional derived Session key id. This api also supports additional mac algorithms.

Parameters

- **session_ctx** – **[in]** The session context
- **objectID** – **[in]** HMAC key object id
- **salt** – **[in]** Salt data
- **saltLen** – **[in]** Salt length
- **saltID** – **[in]** Object id with salt data
- **count** – **[in]** Iteration count
- **macAlgo** – **[in]** MAC Algorithm
- **requestedLen** – **[in]** Requested derived session key length
- **derivedSessionKeyID** – **[inout]** HMAC object id to store output derived session key
- **derivedSessionKey** – **[inout]** Buffer to store derived session key on host
- **pderivedSessionKeyLen** – **[inout]** DerivedSessionKey buffer length

10.15.4.146 Function Se05x_API_ReadCryptoObjectList

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.146.1 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 Crypto Context.

Command to Applet

Field	Value	Description
CLA	0x80	
INS	INS_READ	See SE05x_INS_t
P1	P1_CRYPTO_OBJ	See SE05x_P1_t
P2	P2_LIST	See SE05x_P2_t
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 Crypto Context 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

- **session_ctx – [in]** Session Context [0:kSE05x_pSession]
- **idlist – [out]** If more ids are present [0:kSE05x_TAG_1]
- **pidlistLen – [inout]** Length for idlist

10.15.4.147 Function Se05x_API_ReadECCurveList

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.147.1 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 SE05x_INS_t
P1	P1_CURVE	See SE05x_P1_t
P2	P2_LIST	See SE05x_P2_t
Le	0x00	

R-APDU Body

Value	Description
TLV[TAG_1]	Byte array listing all curve identifiers in SE05x_ECCurve_t (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

- **session_ctx – [in]** Session Context [0:kSE05x_pSession]
- **curveList – [out]** [0:kSE05x_TAG_1]
- **pcurveListLen – [inout]** Length for curveList

10.15.4.148 Function Se05x_API_ReadIDList

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.148.1 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 Read IDList command TAG_1=m, response TAG_1=1, TAG_2=second chunk (n entries) thirst 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 SE05x_INS_t
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_LIST	See SE05x_P2_t
Lc	#(Payload)	
	TLV[TAG_1]	2-byte offset
	TLV[TAG_2]	1-byte type filter: 1 byte from SE05x_SecObjTyp_t 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

- **session_ctx – [in]** Session Context [0:kSE05x_pSession]
- **outputOffset – [in]** output offset [1:kSE05x_TAG_1]
- **filter – [in]** filter [2:kSE05x_TAG_2]
- **pmore – [out]** If more ids are present [0:kSE05x_TAG_1]
- **idlist – [out]** Byte array containing 4-byte identifiers [1:kSE05x_TAG_2]
- **pidlistLen – [inout]** Length for idlist

10.15.4.149 Function Se05x_API_ReadObject

- Defined in [File se05x_APDU/apis.h](#)

10.15.4.149.1 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 SE05x_INS_t , in addition to INS_READ, users can set the INS_ATTEST flag. In that case, attestation applies.
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_DEFAULT	See SE05x_P2_t
Lc	#(Payload)	Payload Length.
	TLV[TAG_1]	4-byte object identifier
	TLV[TAG_2]	2-byte offset [Optional: default 0] [Conditional: only when the object is a Binary File object]
	TLV[TAG_3]	2-byte length [Optional: default 0] [Conditional: only when the object is a Binary File object]
	TLV[TAG_4]	1-byte SE05x_RSAKeyComponent_t : 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

- **session_ctx** – **[in]** Session Context [0:kSE05x_pSession]
- **objectID** – **[in]** object id [1:kSE05x_TAG_1]
- **offset** – **[in]** offset [2:kSE05x_TAG_2]
- **length** – **[in]** length [3:kSE05x_TAG_3]
- **data** – **[out]** [0:kSE05x_TAG_1]
- **pdataLen** – **[inout]** Length for data

10.15.4.150 Function Se05x_API_ReadObject_W_Attst

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.150.1 Function Documentation

smStatus_t Se05x_API_ReadObject_W_Attst(*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_Attst

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 SE05x_INS_t , in addition to INS_READ, users can set the INS_ATTEST flag. In that case, attestation applies.
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_DEFAULT	See SE05x_P2_t
Lc	#(Payload)	Payload Length.
	TLV[TAG_1]	4-byte object identifier
	TLV[TAG_2]	2-byte offset [Optional: default 0] [Conditional: only when the object is a Binary File object]
	TLV[TAG_3]	2-byte length [Optional: default 0] [Conditional: only when the object is a Binary File object]
	TLV[TAG_4]	1-byte SE05x_RSAKeyComponent_t : either RSA_COMP_MOD or RSA_COMP_PUB_EXP. [Optional] [Conditional: only for RSA key components]
	TLV[TAG_5]	4-byte attestation object identifier. [Optional] [Conditional: only when INS_ATTEST is set]

Field	Value	Description
	TLV[TAG_6]	1-byte SE05x_AttestationAlgo_t [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	

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].

Parameters

- **session_ctx** – [in] The session context
- **objectID** – [in] The object id
- **offset** – [in] The offset
- **length** – [in] The length
- **attestID** – [in] The attest id
- **attestAlgo** – [in] The attest algorithm
- **random** – [in] The random
- **randomLen** – [in] The random length
- **data** – The data
- **pdataLen** – The pdata length
- **attribute** – The attribute
- **pattributeLen** – The pattribute length
- **pTimeStamp** – The ptime stamp
- **outrandom** – The chip identifier
- **poutrandomLen** – The pchip identifier length
- **chipId** – The chip identifier
- **pchipIdLen** – The pchip identifier length
- **signature** – The signature
- **psignatureLen** – The psignature length

Returns

The sm status.

10.15.4.151 Function Se05x_API_ReadObject_W_Attst_V2

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.151.1 Function Documentation

smStatus_t Se05x_API_ReadObject_W_Attst_V2(*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* *chipId, *size_t* *pchipIdLen, *uint8_t* *pCmd, *size_t* *pCmdLen, *uint8_t* *pObj, *size_t* *pObjLen, *uint8_t* *signature, *size_t* *psignatureLen)

Se05x_API_ReadObject_W_Attst_V2

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	

Field	Value	Description
INS	INS_READ	See SE05x_INS_t , in addition to INS_READ, users can set the INS_ATTEST flag. In that case, attestation applies.
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_DEFAULT	See SE05x_P2_t
Lc	#(Payload)	Payload Length.
	TLV[TAG_1]	4-byte object identifier
	TLV[TAG_2]	2-byte offset [Optional: default 0] [Conditional: only when the object is a Binary File object]
	TLV[TAG_3]	2-byte length [Optional: default 0] [Conditional: only when the object is a Binary File object]
	TLV[TAG_4]	1-byte SE05x_RSAKeyComponent_t : either RSA_COMP_MOD or RSA_COMP_PUB_EXP. [Optional] [Conditional: only for RSA key components]
	TLV[TAG_5]	4-byte attestation object identifier. [Optional] [Conditional: only when INS_ATTEST is set]
	TLV[TAG_6]	1-byte SE05x_AttestationAlgo_t [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	

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 Object AttributesRef).
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 Object AttributesRef).
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].

Parameters

- **session_ctx** – [in] The session context
- **objectID** – [in] The object id
- **offset** – [in] The offset
- **length** – [in] The length
- **attestID** – [in] The attest id
- **attestAlgo** – [in] The attest algorithm
- **random** – [in] The random
- **randomLen** – [in] The random length
- **data** – The data
- **pdataLen** – The pdata length
- **attribute** – The attribute
- **pattributeLen** – The pattribute length
- **pTimeStamp** – The ptime stamp
- **chipId** – The chip identifier
- **pchipIdLen** – The pchip identifier length
- **pCmd** – The pointer to command buffer
- **pCmdLen** – The length of cmd buffer
- **pObj** – The secure object
- **pObjLen** – The length of secure object
- **signature** – The signature
- **psignatureLen** – The psignature length

Returns

The sm status.

10.15.4.152 Function Se05x_API_ReadObjectAttributes

- Defined in [File se05x_04_xx_APDU_apis.h](#)

10.15.4.152.1 Function Documentation

smStatus_t **Se05x_API_ReadObjectAttributes**(**pSe05xSession_t** session_ctx, **uint32_t** objectID, **uint8_t** *data, **size_t** *pdataLen)

Se05x_API_ReadObjectAttributes

Parameters

- **session_ctx** – [in] Session Context [0:kSE05x_pSession]
- **objectID** – [in] object id [1:kSE05x_TAG_1]
- **data** – [out] [0:kSE05x_TAG_2]
- **pdataLen** – [inout] Length for data

10.15.4.153 Function Se05x_API_ReadObjectAttributes_W_Attst_V2

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.153.1 Function Documentation

smStatus_t Se05x_API_ReadObjectAttributes_W_Attst_V2(*pSe05xSession_t* session_ctx, *uint32_t* objectID, *uint32_t* attestID, *SE05x_AttestationAlgo_t* attestAlgo, *const uint8_t* *random, *size_t* randomLen, *uint8_t* *pCmdapdu, *size_t* *pCmdapduLen, *uint8_t* *data, *size_t* *pdataLen, *SE05x_TimeStamp_t* *pTimeStamp, *uint8_t* *obj, *size_t* *pobjLen, *uint8_t* *chipId, *size_t* *pchipIdLen, *uint8_t* *signature, *size_t* *psignatureLen)

Se05x_API_ReadObjectAttributes_W_Attst_V2

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 identifiersRef
Object type	1	One of SecureObjectType
Authentication attribute	1	One of SetIndicatorRef
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 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.

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

Attribute	Size (bytes)	Description
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 SE05x_INS_t , in addition to INS_READ, users can set the INS_ATTEST flag. In that case, attestation applies.
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_ATTRIBUTES	See SE05x_P2_t
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]	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.

Parameters

- **session_ctx** – [in] The session context
- **objectID** – [in] The object id
- **attestID** – [in] The attest id
- **attestAlgo** – [in] The attest algorithm
- **random** – [in] The random
- **randomLen** – [in] The random length
- **pCmdapdu** – The pointer to command APDU
- **pCmdapduLen** – The pointer to command APDU length
- **data** – The data
- **pdataLen** – The pdata length
- **ptimeStamp** – The ptime stamp
- **chipId** – The chip identifier
- **obj** – The pointer to secure object buffer
- **pobjLen** – The pointer to secure object buffer length
- **pchipIdLen** – The pchip identifier length
- **signature** – The signature
- **psignatureLen** – The psignature length

Returns

The sm status.

10.15.4.154 Function Se05x_API_ReadRSA

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.154.1 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

- **session_ctx** – [in] Session Context [0:kSE05x_pSession]
- **objectID** – [in] object id [1:kSE05x_TAG_1]
- **offset** – [in] offset [2:kSE05x_TAG_2]
- **length** – [in] length [3:kSE05x_TAG_3]
- **rsa_key_comp** – [in] rsa_key_comp [4:kSE05x_TAG_4]
- **data** – [out] [0:kSE05x_TAG_1]
- **pdataLen** – [inout] Length for data

10.15.4.155 Function Se05x_API_ReadRSA_W_Attst_V2

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.155.1 Function Documentation

`smStatus_t Se05x_API_ReadRSA_W_Attst_V2(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 *chipId, size_t *pchipIdLen, uint8_t *pCmd, size_t *pCmdLen, uint8_t *pObj, size_t *pObjLen, uint8_t *signature, size_t *psignatureLen)`

Se05x_API_ReadRSA_W_Attst_V2

See [Se05x_API_ReadObject_W_Attst](#)

Parameters

- `session_ctx` – [in] The session context
- `objectID` – [in] The object id
- `offset` – [in] The offset
- `length` – [in] The length
- `rsa_key_comp` – [in] The rsa key component
- `attestID` – [in] The attest id
- `attestAlgo` – [in] The attest algorithm
- `random` – [in] The random
- `randomLen` – [in] The random length
- `data` – The data
- `pdataLen` – The pdata length
- `attribute` – The attribute
- `pattributeLen` – The pattribute length
- `ptimeStamp` – The ptime stamp
- `chipId` – The chip identifier
- `pchipIdLen` – The pchip identifier length
- `pCmd` – The pointer to command buffer
- `pCmdLen` – The pointer to command buffer length
- `pObj` – The pointer to secure object buffer
- `pObjLen` – The pointer to secure object buffer length
- `signature` – The signature
- `psignatureLen` – The psignature length

Returns

The sm status.

10.15.4.156 Function Se05x_API_ReadSize

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.156.1 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 SE05x_INS_t
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_SIZE	See SE05x_P2_t
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.

Parameters

- **session_ctx – [in]** The session context
- **objectID – [in]** The object id
- **psize –** The psize

Returns

The sm status.

10.15.4.157 Function Se05x_API_ReadState

- Defined in [File se05x_04_xx_APDU_apis.h](#)

10.15.4.157.1 Function Documentation

[smStatus_t](#) **Se05x_API_ReadState(*pSe05xSession_t* session_ctx, *uint8_t* *pstateValues, *size_t* *pstateValuesLen)**

Se05x_API_ReadState

Command to Applet

R-APDU Body

NA

R-APDU Trailer

Parameters

- **session_ctx** – [in] Session Context [0:kSE05x_pSession]
- **pstateValues** – [out] [1:kSE05x_TAG_1]
- **pstateValuesLen** – [out] Length of pstateValues

10.15.4.158 Function Se05x_API_ReadType

- Defined in [File se05x_APDU/apis.h](#)

10.15.4.158.1 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 SE05x_INS_t
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_TYPE	See SE05x_P2_t
Lc	#(Payload)	
	TLV[TAG_1]	4-byte object identifier.
Le	0x00	

R-APDU Body

Value	Description
TLV[TAG_1]	Type of the Secure Object: one of SE05x_SecObjTyp_t
TLV[TAG_2]	TransientIndicatorRef

R-APDU Trailer

SW	Description
SW_NO_ERROR	Data is returned successfully.

Parameters

- **session_ctx** – [in] The session context
- **objectID** – [in] The object id
- **ptype** – The ptype
- **pisTransient** – The pis transient
- **attestation_type** – [in] The attestation type

Returns

The sm status.

10.15.4.159 Function Se05x_API_RefreshSession

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.159.1 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 SE05x_INS_t
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_SESSION_REFRESH	See SE05x_P2_t
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

- `session_ctx` – [in] Session Context [0:kSE05x_pSession]
- `policy` – [in] policy [1:kSE05x_TAG_POLICY]

10.15.4.160 Function Se05x_API_RSADecrypt

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.160.1 Function Documentation

`smStatus_t Se05x_API_RSADecrypt(pSe05xSession_t session_ctx, uint32_t objectID, SE05x_RSAEncryptionAlgo_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

Field	Value	Description
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_RSAEncryptionAlgo_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

- **session_ctx** – [in] Session Context [0:kSE05x_pSession]
- **objectID** – [in] objectID [1:kSE05x_TAG_1]
- **rsaEncryptionAlgo** – [in] rsaEncryptionAlgo [2:kSE05x_TAG_2]
- **inputData** – [in] inputData [3:kSE05x_TAG_3]
- **inputDataLen** – [in] Length of inputData
- **decryptedData** – [out] [0:kSE05x_TAG_1]
- **pdecryptedDataLen** – [inout] Length for decryptedData

10.15.4.161 Function Se05x_API_RSAEncrypt

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.161.1 Function Documentation

smStatus_t Se05x_API_RSAEncrypt(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_RSAEncrypt

The RSAEncrypt command encrypts data.

Command to Applet

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	SE05x_INS_t
P1	P1_RSA	See SE05x_P1_t
P2	P2_ENCRYPT_ONESHOT	See SE05x_P2_t

Field	Value	Description
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

- **session_ctx – [in]** Session Context [0:kSE05x_pSession]
- **objectID – [in]** objectID [1:kSE05x_TAG_1]
- **rsaEncryptionAlgo – [in]** rsaEncryptionAlgo [2:kSE05x_TAG_2]
- **inputData – [in]** inputData [3:kSE05x_TAG_3]
- **inputDataLen – [in]** Length of inputData
- **encryptedData – [out]** [0:kSE05x_TAG_1]
- **pencryptedDataLen – [inout]** Length for encryptedData

10.15.4.162 Function Se05x_API_RSASign

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.162.1 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

Name	Value	Description
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_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 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

- **session_ctx – [in]** Session Context [0:kSE05x_pSession]
- **objectId – [in]** objectID [1:kSE05x_TAG_1]
- **rsaSigningAlgo – [in]** rsaSigningAlgo [2:kSE05x_TAG_2]
- **inputData – [in]** inputData [3:kSE05x_TAG_3]
- **inputDataLen – [in]** Length of inputData
- **signature – [out]** [0:kSE05x_TAG_1]
- **psignatureLen – [inout]** Length for signature

10.15.4.163 Function Se05x_API_RSASign_WithSalt

- Defined in [File se05x_04_xx_APDU_apis.h](#)

10.15.4.163.1 Function Documentation

smStatus_t Se05x_API_RSASign_WithSalt([pSe05xSession_t](#) session_ctx, uint32_t objectId, [SE05x_RSASignatureAlgo_t](#) rsaSigningAlgo, const uint8_t *inputData, size_t inputDataLen, uint16_t saltLen, uint8_t *signature, size_t *psignatureLen)

Se05x_API_RSASign_WithSalt

See [Se05x_API_RSASign](#). Se05x_API_RSASign API with salt lenght as extra input.

10.15.4.164 Function Se05x_API_RSAVerify

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.164.1 Function Documentation

**smStatus_t Se05x_API_RSAVerify(pSe05xSession_t session_ctx, uint32_t objectID, SE05x_RSASignatureAlgo_t rsa
SigningAlgo, const uint8_t *inputData, size_t inputDataLen, const uint8_t *signature, size_t signatureLen, SE05x_Result_t
*presult)**

Se05x_API_RSAVerify

The RSAVerify 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 RSAVerify 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)	
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

- **session_ctx** – [in] Session Context [0:kSE05x_pSession]
- **objectID** – [in] objectID [1:kSE05x_TAG_1]
- **rsaSigningAlgo** – [in] rsaSigningAlgo [2:kSE05x_TAG_2]
- **inputData** – [in] inputData [3:kSE05x_TAG_3]
- **inputDataLen** – [in] Length of inputData
- **signature** – [in] signature [4:kSE05x_TAG_5]
- **signatureLen** – [in] Length of signature
- **presult** – [out] [0:kSE05x_TAG_1]

10.15.4.165 Function Se05x_API_RSAVerify_WithSalt

- Defined in [File se05x_04_xx_APDU_apis.h](#)

10.15.4.165.1 Function Documentation

smStatus_t Se05x_API_RSAVerify_WithSalt(*pSe05xSession_t* session_ctx, *uint32_t* objectID, *SE05x_RSASignatureAlgo_t* rsaSigningAlgo, *const uint8_t* **inputData*, *size_t* *inputDataLen*, *uint16_t* *saltLen*, *const uint8_t* **signature*, *size_t* *signatureLen*, *SE05x_Result_t* **presult*)

Se05x_API_RSAVerify_WithSalt

See [Se05x_API_RSAVerify](#). Se05x_API_RSASign API with salt lenght as extra input.

10.15.4.166 Function Se05x_API_SendCardManagerCmd

- Defined in [File se05x_04_xx_APDU_apis.h](#)

10.15.4.166.1 Function Documentation

smStatus_t Se05x_API_SendCardManagerCmd(*pSe05xSession_t* session_ctx, *uint8_t* **pCmdData*, *size_t* *cmdDataLen*, *uint8_t* **pOutputData*, *size_t* **pOutputDataLen*)

Se05x_API_SendCardManagerCmd

Sends a command to the Card Manager.

This APDU will send command to Card Manager

Command to Card Manager

Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See SE05x_INS_t
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_CM_COMMAND	See SE05x_P2_t
Lc	#(Payload)	Payload length
Payload	TLV[TAG_1]	APDU to be sent to the Card Manager.
Le	0x00	Expected response length

R-APDU Body

Value	Description
TLV[TAG_1]	Byte array containing the Card Manager response.

R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

Parameters

- **session_ctx – [in]** The session context
- **pCmdData – [in]** The command input data
- **cmdDataLen – [in]** The command input data length
- **pOutputData – [out]** The response data
- **pOutputDataLen – [out]** The response data length

Returns

The sm status.

10.15.4.167 Function Se05x_API_SetAppletFeatures

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.167.1 Function Documentation

[**smStatus_t Se05x_API_SetAppletFeatures\(*pSe05xSession_t session_ctx, pSe05xAppletFeatures_t appletVariant*\)**](#)

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 SE05x_INS_t
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_VARIANT	See SE05x_P2_t
Lc	#(Payload)	Payload length
Payload	TLV[TAG_1]	2-byte Variant from SE05x_AppletConfig_t

R-APDU Body

NA

R-APDU Trailer

Parameters

- **session_ctx – [in]** Session Context [0:kSE05x_pSession]
- **appletVariant – [in]** Applet variant [1:kSE05x_TAG_1]

10.15.4.168 Function Se05x_API_SetCounterValue

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.168.1 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

- session_ctx** – [in] Session Context [0:kSE05x_pSession]
- objectID** – [in] object id [1:kSE05x_TAG_1]
- size** – [in] size [3:kSE05x_TAG_2]
- value** – [in] value [4:kSE05x_TAG_3]

10.15.4.169 Function Se05x_API_SetECCurveParam

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.169.1 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 SE05x_INS_t
P1	P1_CURVE	See SE05x_P1_t
P2	P2_PARAM	See SE05x_P2_t
Lc	#(Payload)	
	TLV[TAG_1]	1-byte curve identifier, from SE05x_ECCurve_t
	TLV[TAG_2]	1-byte SE05x_ECCurveParam_t
	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

- **session_ctx – [in]** Session Context [0:kSE05x_pSession]
- **curveID – [in]** curve id [1:kSE05x_TAG_1]
- **ecCurveParam – [in]** ecCurveParam [2:kSE05x_TAG_2]
- **inputData – [in]** inputData [3:kSE05x_TAG_3]
- **inputDataLen – [in]** Length of inputData

10.15.4.170 Function Se05x_API_SetLockState

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.170.1 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.

PERSISTENT_LOCK	TRANSIENT_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 SE05x_INS_t
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_TRANSPORT	See SE05x_P2_t
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

- **session_ctx** – [in] Session Context [0:kSE05x_pSession]
- **lockIndicator** – [in] lock indicator [1:kSE05x_TAG_1]
- **lockState** – [in] lock state [2:kSE05x_TAG_2]

10.15.4.171 Function Se05x_API_SetPlatformSCPRequest

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.171.1 Function Documentation

smStatus_t Se05x_API_SetPlatformSCPRequest([pSe05xSession_t](#) session_ctx, [SE05x_PlatformSCPRequest_t](#) platform SCPRequest)

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 RESERVED_ID_PLATFORM_SCOP as authentication object.

Note that the default state is SCP_NOT_REQUIRED.

Command to Applet

Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See SE05x_INS_t
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_SCOP	See SE05x_P2_t
Lc	#(Payload)	
Payload	TLV[TAG_1]	1-byte SE05x_PlatformSCPRequest_t
Le		

R-APDU Body

NA

R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

Parameters

- **session_ctx** – [in] Session Context [0:kSE05x_pSession]
- **platformSCPRequest** – [in] platf scp req [1:kSE05x_TAG_1]

10.15.4.172 Function Se05x_API_TLSCalculatePreMasterSecret

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.172.1 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 SE05x_INS_t
P1	P1_TLS	See SE05x_P1_t
P2	P2_PMS	See SE05x_P2_t
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

- **session_ctx** – [in] Session Context [0:kSE05x_pSession]
- **keyPairId** – [in] keyPairId [1:kSE05x_TAG_1]
- **pskId** – [in] pskId [2:kSE05x_TAG_2]
- **hmacKeyId** – [in] hmacKeyId [3:kSE05x_TAG_3]
- **inputData** – [in] inputData [4:kSE05x_TAG_4]
- **inputDataLen** – [in] Length of inputData

10.15.4.173 Function Se05x_API_TLSCalculateRsaPreMasterSecret

- Defined in [File se05x_04_xx_APDU_apis.h](#)

10.15.4.173.1 Function Documentation

smStatus_t **Se05x_API_TLSCalculateRsaPreMasterSecret**(*pSe05xSession_t session_ctx, uint32_t keyPairId, uint32_t pskId, uint32_t hmacKeyId, const uint8_t *inputData, size_t inputDataLen, const uint8_t *clientVersion, size_t clientVersionLen*)

Se05x_API_TLSCalculateRsaPreMasterSecret

Parameters

- **session_ctx** – [in] Session Context[0:kSE05x_pSession]
- **keyPairId** – [in] keyPairId[1:kSE05x_TAG_1]
- **pskId** – [in] pskId[2:kSE05x_TAG_2]
- **hmacKeyId** – [in] hmacKeyId[3:kSE05x_TAG_3]
- **inputData** – [in] inputData[4:kSE05x_TAG_4]
- **inputDataLen** – [in] Length of inputData
- **clientVersion** – [in] client version[6:kSE05x_TAG_6]
- **clientVersionLen** – [in] Length of client version

10.15.4.174 Function Se05x_API_TLSGenerateRandom

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.174.1 Function Documentation

smStatus_t **Se05x_API_TLSGenerateRandom**(*pSe05xSession_t session_ctx, uint8_t *randomValue, size_t *randomValueLen*)

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_CRYPTO	See SE05x_INS_t
P1	P1_TLS	See SE05x_P1_t

Field	Value	Description
P2	P2_RANDOM	See SE05x_P2_t
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

- **session_ctx** – [in] Session Context [0:kSE05x_pSession]
- **randomValue** – [out] [0:kSE05x_TAG_1]
- **prandomValueLen** – [inout] Length for randomValue

10.15.4.175 Function Se05x_API_TLSPerformPRF

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.175.1 Function Documentation

smStatus_t Se05x_API_TLSPerformPRF(*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_TLSPerformPRFTYPE_t tlspf*)

Se05x_API_TLSPerformPRF

The command TLSPerformPRF 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 SE05x_INS_t
P1	P1_TLS	See SE05x_P1_t
P2	See description above.	See SE05x_P2_t
Lc	#(Payload)	
	TLV[TAG_1]	4-byte HMACKey identifier.
	TLV[TAG_2]	1-byte SE05x_DigestMode_t , 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.

Parameters

- **session_ctx – [in]** The session context
- **objectId – [in]** The object id
- **digestAlgo – [in]** The digest algorithm
- **label – [in]** The label
- **labelLen – [in]** The label length
- **random – [in]** The random
- **randomLen – [in]** The random length
- **reqLen – [in]** The request length
- **outputData –** The output data
- **poutputDataLen –** The poutput data length
- **tlsprf – [in]** The tlsprf

Returns

The sm status.

10.15.4.176 Function Se05x_API_TriggerSelfTest

- Defined in [File se05x_04_xx_APDU_apis.h](#)

10.15.4.176.1 Function Documentation

`smStatus_t Se05x_API_TriggerSelfTest(pSe05xSession_t session_ctx, SE05x_HealthCheckMode_t healthCheckMode, uint8_t *result)`

Se05x_API_TriggerSelfTest

Trigger a system health check for the system. When calling this command, a self-test is triggered in the operating system. When the test fails, the device might not respond with a R-APDU as the chip is reset. If HealthCheckMode is set to HCM_FIPS, the test will only work if the device is running in FIPS approved mode of operation.

Command to Applet

Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See SE05x_INS_t . In addition to INS_CRYPTO, users can set the INS_ATTEST flag. In that case, attestation applies.
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_SANITY	See SE05x_P2_t
Lc	#(Payload)	Payload length
Payload	TLV[TAG_1]	2-byte value from HealthCheckMode
Le	0x00	2-byte response + attested data (if INS_ATTEST is set).

R-APDU Body

Value	Description
TLV[TAG_1]	TLV containing 1-byte Result.

R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

Parameters

- session_ctx** – [in] The session context
- healthCheckMode** – [in] The health check mode
- result** – The result of Self Test

Returns

The sm status.

10.15.4.177 Function Se05x_API_TriggerSelfTest_W_Attst

- Defined in [File se05x_04_xx_APDU_apis.h](#)

10.15.4.177.1 Function Documentation

`smStatus_t Se05x_API_TriggerSelfTest_W_Attst(pSe05xSession_t session_ctx, SE05x_HealthCheckMode_t healthCheckMode, uint32_t attestID, SE05x_AttestationAlgo_t attestAlgo, const uint8_t *random, size_t randomLen, uint8_t *result, SE05x_TimeStamp_t *pTimeStamp, uint8_t *chipId, size_t *pchipIdLen, uint8_t *signature, size_t *psignatureLen, uint8_t *pObjectSize, size_t *pObjectSizeLen, uint8_t *pCmd, size_t *pCmdLen)`

Se05x_API_TriggerSelfTest_W_Attst

Trigger a system health check for the system. When calling this command, a self-test is triggered in the operating system. When the test fails, the device might not respond with a R-APDU as the chip is reset. If HealthCheckMode is set to HCM_FIPS, the test will only work if the device is running in FIPS approved mode of operation.

Command to Applet

Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See SE05x_INS_t . In addition to INS_CRYPTO, users can set the INS_ATTEST flag. In that case, attestation applies.
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_SANITY	See SE05x_P2_t
Lc	#(Payload)	Payload length
Payload	TLV[TAG_1]	2-byte value from HealthCheckMode
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	2-byte response + attested data (if INS_ATTEST is set).

R-APDU Body

Value	Description
TLV[TAG_1]	TLV containing 1-byte Result.
TLV[TAG_3]	TLV containing 12-byte timestamp [Conditional: only when C-APDU contains INS_ATTEST]
TLV[TAG_4]	TLV containing 16-byte freshness (random) [Conditional: only when C-APDU contains INS_ATTEST]
TLV[TAG_5]	TLV containing 18-byte chip unique ID [Conditional: only when C-APDU contains INS_ATTEST]
TLV[TAG_6]	TLV containing signature over the concatenated values of TLV[TAG_1], TLV[TAG_3], TLV[TAG_4] and TLV[TAG_5]. [Conditional: only when C-APDU contains INS_ATTEST]

R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

Parameters

- **session_ctx** – [in] The session context
- **healthCheckMode** – [in] The health check mode
- **attestID** – [in] The attest id
- **attestAlgo** – [in] The attest algorithm
- **random** – [in] The random
- **randomLen** – [in] The random length
- **result** – The result of Self Test
- **pTimeStamp** – The ptime stamp
- **chipId** – The chip identifier
- **pchipIdLen** – The pchip identifier length
- **signature** – The signature
- **psignatureLen** – The psignature length
- **pObjectSize** – The pObjectSize
- **pObjectSizeLen** – The pObjectSizeLen length
- **pCmd** – The pCmd
- **pCmdLen** – The pCmd length

Returns

The sm status.

10.15.4.178 Function Se05x_API_UpdateBinary_Ver

- Defined in [File se05x_04_xx_APDU_apis.h](#)

10.15.4.178.1 Function Documentation

`smStatus_t Se05x_API_UpdateBinary_Ver(pSe05xSession_t session_ctx, pSe05xPolicy_t policy, uint32_t objectID, uint16_t offset, uint16_t length, const uint8_t *inputData, size_t inputDataLen, uint32_t version)`

Se05x_API_UpdateBinary_Ver

See [Se05x_API_WriteBinary](#). Also allows to set key version (4 bytes). Called to update the value of already existing object. If policy is passed, it should match with existing policy on object.

10.15.4.179 Function Se05x_API_UpdateCounter

- Defined in [File se05x_04_xx_APDU_apis.h](#)

10.15.4.179.1 Function Documentation

`smStatus_t Se05x_API_UpdateCounter(pSe05xSession_t session_ctx, pSe05xPolicy_t policy, uint32_t objectID, uint16_t size, uint64_t value)`

Se05x_API_UpdateCounter

See [Se05x_API_SetCounterValue](#). Called to update the value of already existing object. If policy is passed, it should match with existing policy on object.

10.15.4.180 Function Se05x_API_UpdateECKey_Ver

- Defined in [File se05x_04_xx_APDU_apis.h](#)

10.15.4.180.1 Function Documentation

`smStatus_t Se05x_API_UpdateECKey_Ver(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, uint32_t version)`

Se05x_API_UpdateECKey_Ver

See [Se05x_API_WriteECKey](#). Also allows to set key version (4 bytes). Called to update the value of already existing object. If policy is passed, it should match with existing policy on object.

10.15.4.181 Function Se05x_API_UpdatePCR

- Defined in [File se05x_04_xx_APDU_apis.h](#)

10.15.4.181.1 Function Documentation

`smStatus_t Se05x_API_UpdatePCR(pSe05xSession_t session_ctx, pSe05xPolicy_t policy, uint32_t pcrlID, const uint8_t *inputData, size_t inputDataLen)`

Se05x_API_UpdatePCR

See [Se05x_API_WritePCR](#). Called to update the value of already existing object. If policy is passed, it should match with existing policy on object.

10.15.4.182 Function Se05x_API_UpdateRSAKey_Ver

- Defined in [File se05x_04_xx_APDU_apis.h](#)

10.15.4.182.1 Function Documentation

`smStatus_t Se05x_API_UpdateRSAKey_Ver(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 ins_type, const SE05x_KeyPart_t key_part, const SE05x_RSAKeyFormat_t rsa_format, uint32_t version)`

Se05x_API_UpdateRSAKey_Ver

See [Se05x_API_WriteRSAKey](#). Also allows to set key version (4 bytes). Called to update the value of already existing object. If policy is passed, it should match with existing policy on object.

10.15.4.183 Function Se05x_API_UpdateSymmKey_Ver

- Defined in [File se05x_04_xx_APDU_apis.h](#)

10.15.4.183.1 Function Documentation

`smStatus_t Se05x_API_UpdateSymmKey_Ver(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, uint32_t version)`

Se05x_API_UpdateSymmKey_Ver

See [Se05x_API_WriteSymmKey](#). Also allows to set key version (4 bytes). Called to update the value of already existing object. If policy is passed, it should match with existing policy on object.

10.15.4.184 Function Se05x_API_VerifySessionUserID

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.184.1 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 SE05x_INS_t
P1	P1_DEFAULT	See SE05x_P1_t
P2	P2_SESSION_USERID	See SE05x_P2_t
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

- `session_ctx` – [in] Session Context [0:kSE05x_pSession]
- `userId` – [in] userId [1:kSE05x_TAG_1]
- `userIdLen` – [in] Length of userId

10.15.4.185 Function Se05x_API_WriteBinary

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.185.1 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 SE05x_P1_t
P2	P2_DEFAULT	See SE05x_P2_t
Payload	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 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

- **session_ctx** – **[in]** Session Context [0:kSE05x_pSession]
- **policy** – **[in]** policy [1:kSE05x_TAG_POLICY]
- **objectID** – **[in]** object id [2:kSE05x_TAG_1]
- **offset** – **[in]** offset [3:kSE05x_TAG_2]
- **length** – **[in]** length [4:kSE05x_TAG_3]
- **inputData** – **[in]** input data [5:kSE05x_TAG_4]
- **inputDataLen** – **[in]** Length of inputData

10.15.4.186 Function Se05x_API_WriteBinary_Ver

- Defined in [File se05x_04_xx_APDU_apis.h](#)

10.15.4.186.1 Function Documentation

smStatus_t Se05x_API_WriteBinary_Ver(pSe05xSession_t session_ctx, pSe05xPolicy_t policy, uint32_t objectID, uint16_t offset, uint16_t length, const uint8_t *inputData, size_t inputDataLen, uint32_t version)

Se05x_API_WriteBinary_Ver

See [Se05x_API_WriteBinary](#). Also allows to set key version (4 bytes).

10.15.4.187 Function Se05x_API_WriteECKey

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.187.1 Function Documentation

smStatus_t Se05x_API_WriteECKey(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_WriteECKey

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
Payload	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_MAX_ATTEMPTS]	2-byte maximum 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 ECKeyRef) [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 ECKeyRef) [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]

Parameters

- **session_ctx – [in]** The session context
- **policy – [in]** The policy
- **maxAttempt – [in]** The maximum attempt
- **objectID – [in]** The object id
- **curveID – [in]** The curve id
- **privKey – [in]** The priv key
- **privKeyLen – [in]** The priv key length
- **pubKey – [in]** The pub key
- **pubKeyLen – [in]** The pub key length
- **ins_type – [in]** The insert type
- **key_part – [in]** The key part

Returns

The sm status.

10.15.4.188 Function Se05x_API_WriteECKey_Ver

- Defined in [File se05x_04_xx_APDU_apis.h](#)

10.15.4.188.1 Function Documentation

```
smStatus_t Se05x_API_WriteECKey_Ver(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 *pub
Key, size_t pubKeyLen, const SE05x_INS_t ins_type, const SE05x_KeyPart_t key_part, uint32_t version)
```

Se05x_API_WriteECKey_Ver

See [Se05x_API_WriteECKey](#). Also allows to set key version (4 bytes).

10.15.4.189 Function Se05x_API_WritePCR

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.189.1 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 SE05x_P1_t
P2	P2_DEFAULT	See SE05x_P2_t
Payload	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 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

- session_ctx – [in]** Session Context [0:kSE05x_pSession]
- policy – [in]** policy [1:kSE05x_TAG_POLICY]
- pcrID – [in]** object id [2:kSE05x_TAG_1]
- initialValue – [in]** initialValue [3:kSE05x_TAG_2]
- initialValueLen – [in]** Length of initialValue
- inputData – [in]** inputData [4:kSE05x_TAG_3]
- inputDataLen – [in]** Length of inputData

10.15.4.190 Function Se05x_API_WritePCR_WithType

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.190.1 Function Documentation

`smStatus_t Se05x_API_WritePCR_WithType(pSe05xSession_t session_ctx, const SE05x_INS_t ins_type, pSe05xPolicy_t policy, uint32_t pcrlID, const uint8_t *initialValue, size_t initialValueLen, const uint8_t *inputData, size_t inputDataLen)`

Se05x_API_WritePCR_WithType

See [Se05x_API_WritePCR](#)

10.15.4.191 Function Se05x_API_WriteRSAKey

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.191.1 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_RSAKeyFormat_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 RSAKey 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	SE05x_KeyPart_t P1_RSA	See SE05x_P1_t
P2	P2_DEFAULT or P2_RAW	See SE05x_P2_t ; 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.

Field	Value	Description
Payload	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 (SE05x_RSABitLength_t) [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 P1Key Part 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 P1Key Part 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 P1 KeyPart 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]

Parameters

- **session_ctx – [in]** The session context
- **policy – [in]** The policy
- **objectID – [in]** The object id
- **size – [in]** The size
- **p – [in]** The part p
- **pLen – [in]** The p length
- **q – [in]** The quarter
- **qLen – [in]** The quarter length
- **dp – [in]** The part dp
- **dpLen – [in]** The dp length
- **dq – [in]** The part dq
- **dqLen – [in]** The dq length
- **qInv – [in]** The quarter inv
- **qInvLen – [in]** The quarter inv length
- **pubExp – [in]** The pub exponent
- **pubExpLen – [in]** The pub exponent length
- **priv – [in]** The priv
- **privLen – [in]** The priv length
- **pubMod – [in]** The pub modifier
- **pubModLen – [in]** The pub modifier length
- **transient_type – [in]** The transient type
- **key_part – [in]** The key part
- **rsa_format – [in]** The rsa format

Returns

The sm status.

10.15.4.192 Function Se05x_API_WriteRSAKey_Ver

- Defined in [File se05x_04_xx_APDU_apis.h](#)

10.15.4.192.1 Function Documentation

```
smStatus_t Se05x_API_WriteRSAKey_Ver(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 priv  
Len, const uint8_t *pubMod, size_t pubModLen, const SE05x_INS_t ins_type, const SE05x_KeyPart_t key_part, const  
SE05x_RSAKeyFormat_t rsa_format, uint32_t version)
```

Se05x_API_WriteRSAKey_Ver

See [Se05x_API_WriteRSAKey](#). Also allows to set key version (4 bytes).

10.15.4.193 Function Se05x_API_WriteSymmKey

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.193.1 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 SE05x_P1_t
P2	P2_DEFAULT	See SE05x_P2_t
Payload	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_MAX_ATTEMPTS]	2-byte maximum 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]

Parameters

- `session_ctx` – [in] The session context
- `policy` – [in] The policy
- `maxAttempt` – [in] The maximum attempt
- `objectID` – [in] The object id
- `kekID` – [in] The kek id
- `keyValue` – [in] The key value
- `keyValueLen` – [in] The key value length
- `ins_type` – [in] The insert type
- `type` – [in] The type

Returns

The sm status.

10.15.4.194 Function Se05x_API_WriteSymmKey_Ver

- Defined in [File se05x_04_xx_APDU_apis.h](#)

10.15.4.194.1 Function Documentation

`smStatus_t Se05x_API_WriteSymmKey_Ver(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, uint32_t version)`

Se05x_API_WriteSymmKey_Ver

See [Se05x_API_WriteSymmKey](#). Also allows to set key version (4 bytes).

10.15.4.195 Function Se05x_API_WriteSymmKey_Ver_extended

- Defined in [File se05x_04_xx_APDU_apis.h](#)

10.15.4.195.1 Function Documentation

`smStatus_t Se05x_API_WriteSymmKey_Ver_extended(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, uint32_t version, uint16_t min_aead_tag_len)`

Se05x_API_WriteSymmKey_Ver_extended

See [Se05x_API_WriteSymmKey_Ver](#). Extension of Se05x_API_WriteSymmKey_Ver api .Allows to set minimum tag length for AEAD (2 bytes).

10.15.4.196 Function Se05x_API_WriteUserID

- Defined in [File se05x_APDU_apis.h](#)

10.15.4.196.1 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 SE05x_P1_t
P2	P2_DEFAULT	See SE05x_P2_t
	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_MAX_ATTEMPTS]	2-byte maximum 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.

Field	Value	Description
	TLV[TAG_2]	Byte array containing 4 to 16 bytes user identifier value.

Parameters

- **session_ctx – [in]** The session context
- **policy – [in]** The policy
- **maxAttempt – [in]** The maximum attempt
- **objectID – [in]** The object id
- **userId – [in]** The user identifier
- **userIdLen – [in]** The user identifier length
- **attestation_type – [in]** The attestation type

Returns

The sm status.

10.15.4.197 Function se05x_DeCrypt

- Defined in [File se05x_tlv.h](#)

10.15.4.197.1 Function Documentation

```
smStatus_t se05x_DeCrypt(struct Se05xSession *pSessionCtx, size_t cmd_cmacLen, uint8_t *rsp, size_t *rspLength,
uint8_t hasle)
```

10.15.4.198 Function se05x_get_sha_algo

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.198.1 Function Documentation

```
SE05x_DigestMode_t se05x_get_sha_algo(sss_algorithm_t algorithm)
```

Gets the SE05x Digest Mode correspanding to the provided sha algorithm

10.15.4.199 Function se05x_GetAppletVersion

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.4.199.1 Function Documentation

```
uint32_t se05x_GetAppletVersion(void)
```

Returns the applet version compiled by MW

10.15.4.200 Function Se05x_i2c_master_attst_txn

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.4.200.1 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, sss_se05x_attst_comp_data_t *pattest_data, uint8_t *rspbuffer, size_t *rspbufferLen, uint8_t noOftags)`

Se05x_i2c_master_attst_txn.

I2CM Read With Attestation

Parameters

- `sess` – [in] session identifier
- `keyObject` – [in] Keyobject which contains 4 byte attestaion KeyId
- `p` – [inout] Array of structure type capturing a sequence of i2c master cmd/rsp transactions.
- `random_attst` – [in] 16-byte freshness random
- `random_attstLen` – [in] length of freshness random
- `attst_algo` – [in] 1 byte attestationAlgo
- `pattest_data` – [in] Data Related to Attestation process
- `rspbuffer` – [out] The read response
- `rspbufferLen` – [out] Length of the response
- `noOftags` – [in] Amount of structures contained in p

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.

10.15.4.201 Function Se05x_i2c_master_txn

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.4.201.1 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

Parameters

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

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.

10.15.4.202 Function se05x_sssKeyTypeLenToCurveld

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.4.202.1 Function Documentation

```
uint32_t se05x_sssKeyTypeLenToCurveld(sss\_cipher\_type\_t keyType, size\_t keyBits)
```

10.15.4.203 Function se05x_Transform

- Defined in [File se05x_tlv.h](#)

10.15.4.203.1 Function Documentation

```
smStatus\_t se05x_Transform(struct Se05xSession *pSession, const tlvHeader\_t *hdr, uint8\_t *cmdApduBuf, const size\_t cmdApduBufLen, tlvHeader\_t *out_hdr, uint8\_t *txBuf, size\_t *ptxBufLen, uint8\_t hasle)
```

10.15.4.204 Function se05x_Transform_scp

- Defined in [File se05x_tlv.h](#)

10.15.4.204.1 Function Documentation

```
smStatus\_t se05x_Transform_scp(struct Se05xSession *pSession, const tlvHeader\_t *hdr, uint8\_t *cmdApduBuf, const size\_t cmdApduBufLen, tlvHeader\_t *outhdr, uint8\_t *txBuf, size\_t *ptxBufLen, uint8\_t hasle)
```

10.15.4.205 Function sems_lite_agent_init_context

- Defined in [File sems_lite_api.h](#)

10.15.4.205.1 Function Documentation

```
ss\_status\_t sems_lite_agent_init_context(sems\_lite\_agent\_ctx\_t *context, sss\_session\_t *boot_ctx)
```

Initialize SEMS Lite agent context.

This function is used to initialize SEMS Lite agent context.

Parameters

- context** – Pointer to sems lite agent context.
- boot_ctx** – Pointer to sss session context

Return values

- kStatus_SEMS_Lite_Success** – The operation has completed successfully.
- kStatus_SEMS_Lite_Error_General** – The operation has failed.

Returns

Status of the operation

10.15.4.206 Function sems_lite_agent_load_package

- Defined in [File sems_lite_api.h](#)

10.15.4.206.1 Function Documentation

`sems_lite_status_t sems_lite_agent_load_package(sems_lite_agent_ctx_t *context, multicast_package_t *multiPkgBuf)`

Load Applet package.

This function load an available update package on the SE and assure the tearing safe update of the SE.

Note:

More return codes would be added to request host to either retry or install older package.

Parameters

- `context` – Pointer to sems lite agent context.
- `multiPkgBuf` – Pointer to package. It must follow the format defined in [multicast_package_t](#).

Return values

- `kStatus_SEMS_Lite_Success` – The operation has completed successfully.
- `kStatus_SEMS_Lite_ERR_COM` – Communication to SE failed.
- `kStatus_SEMS_Lite_ERR_DoReRun` – Update not completed please provide update package again.
- `kStatus_SEMS_Lite_ERR_NotApplicable` – Update not applicable on this Chip/type.
- `kStatus_SEMS_Lite_ERR_DoRecovery` – Update can not be completed. Please provide recovery package, to roll back to last working version.
- `kStatus_SEMS_Lite_ERR_Fatal` – Unresolvable error. (This category of errors can only appear in testing of an update package, in the case of NXP updates this is testes before by NXP)
- `kStatus_SEMS_Lite_ERR_NotEnoughNVMemory` – Don't have has enough NV memory for the SEMS Lite Script.
- `kStatus_SEMS_Lite_ERR_NotEnoughTransientMemory` – Don't have enough transient memory for the SEMS Lite Script .

Returns

Status of the operation

10.15.4.207 Function `sems_lite_agent_session_close`

- Defined in [File sems_lite_api.h](#)

10.15.4.207.1 Function Documentation

`sss_status_t sems_lite_agent_session_close(sems_lite_agent_ctx_t *context)`

Close the connection to SEMS Lite Applet.

Parameters

`context` – The context

Return values

- `kStatus_SEMS_Lite_Success` – Could close connection 1.
- `kStatus_SEMS_Lite_ERR_General` – Could not close connection 1.

Returns

The api status.

10.15.4.208 Function `sems_lite_agent_session_open`

- Defined in [File sems_lite_api.h](#)

10.15.4.208.1 Function Documentation

`sss_status_t sems_lite_agent_session_open(sems_lite_agent_ctx_t *context)`

Open a Physical connection to SEMS Lite Applet.

Calling this API opens Locical Connection 1 and selecs the SEMS Lite applet.

Parameters

`context` – SEMS Lite Agent Context

Return values

- `kStatus_SEMS_Lite_Success` – Could connect to SEMS Lite Applet.
- `kStatus_SEMS_Lite_ERR_General` – Could not connect to SEMS Lite Applet.

Returns

The api status.

10.15.4.209 Function `sems_lite_check_AppleRecoveryStatus`

- Defined in [File sems_lite_api.h](#)

10.15.4.209.1 Function Documentation

`sss_status_t sems_lite_check_AppleRecoveryStatus(sems_lite_agent_ctx_t *pContext, sems_lite_recovery_status_t *pRecoveryStatus)`

Check Applet Recovery Status.

This API will return the status of applet recovery status

Parameters

- `pContext` – Pointer to sems lite agent context.
- `pRecoveryStatus` – Pointer to recovery status.

Return values

- `kStatus_SSS_Success` – The operation has completed successfully.
- `kStatus_SSS_Fail` – The operation has failed.

Returns

Status of the operation

10.15.4.210 Function `sems_lite_check_AppleUpgradeProgress`

- Defined in [File sems_lite_api.h](#)

10.15.4.210.1 Function Documentation

`sss_status_t sems_lite_check_AppleUpgradeProgress(sems_lite_agent_ctx_t *pContext, sems_lite_upgradeProgress_status_t *pUpgradeStatus)`

Check Applet Upgrade Progress.

This API will return the status of applet upgrade progress status

Parameters

- **pContext** – Pointer to sems lite agent context.
- **pUpgradeStatus** – Pointer to upgrade status.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

10.15.4.211 Function `sems_lite_check_Tear`

- Defined in [File sems_lite_api.h](#)

10.15.4.211.1 Function Documentation

`sss_status_t sems_lite_check_Tear(sems_lite_agent_ctx_t *pContext, sems_lite_tearDown_status_t *pTearStatus)`

Check Tear during script execution.

This API will check whether there was tear during script execution

Parameters

- **pContext** – Pointer to sems lite agent context.
- **pTearStatus** – Pointer to tear status.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

10.15.4.212 Function `sems_lite_get_AgentVersion`

- Defined in [File sems_lite_api.h](#)

10.15.4.212.1 Function Documentation

`sss_status_t sems_lite_get_AgentVersion(uint8_t *pRspBuf, size_t *pRspBufLen)`

Get the SEMS Lite Agent version.

This API will return the SEMS Lite Agent Version no.

Parameters

- **pRspBuf** – Pointer to response Buffer.
- **pRspBufLen** – Pointer to length of the response Buffer

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

10.15.4.213 Function `sems_lite_get_AppletVersion`

- Defined in [File sems_lite_api.h](#)

10.15.4.213.1 Function Documentation

`sss_status_t sems_lite_get_AppletVersion(sems_lite_agent_ctx_t *pContext, uint8_t *pRspBuf, size_t *pRspBufLen)`

Get the Applet version.

This API will return the SEMS Lite Applet Version no.

Parameters

- `pContext` – Pointer to sems lite agent context.
- `pRspBuf` – Pointer to response Buffer.
- `pRspBufLen` – Pointer to length of the response Buffer

Return values

- `kStatus_SSS_Success` – The operation has completed successfully.
- `kStatus_SSS_Fail` – The operation has failed.

Returns

Status of the operation

10.15.4.214 Function `sems_lite_get_available_mem`

- Defined in [File sems_lite_api.h](#)

10.15.4.214.1 Function Documentation

`sss_status_t sems_lite_get_available_mem(sems_lite_agent_ctx_t *pContext, uint8_t *pAvailableMem)`

{ function_description }

Parameters

- `pContext` – The context
- `pAvailableMem` – Available Memory Space Information

Returns

The sss status.

10.15.4.215 Function `sems_lite_get_CA_identifier`

- Defined in [File sems_lite_api.h](#)

10.15.4.215.1 Function Documentation

`sss_status_t sems_lite_get_CA_identifier(sems_lite_agent_ctx_t *pContext, uint8_t *pRspBuf, size_t *pRspBufLen)`

Get the CA Identifier.

This API will return the CA Identifier.

Parameters

- `pContext` – Pointer to sems lite agent context.
- `pRspBuf` – Pointer to response Buffer.
- `pRspBufLen` – Pointer to length of the response Buffer

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

10.15.4.216 Function `sems_lite_get_ENCIdentifier`

- Defined in [File sems_lite_api.h](#)

10.15.4.216.1 Function Documentation

`sss_status_t sems_lite_get_ENCIdentifier(sems_lite_agent_ctx_t *pContext, uint8_t *pRspBuf, size_t *pRspBufLen)`

Get the ENC Identifier.

This API will return the ENC Identifier.

Parameters

- **pContext** – Pointer to sems lite agent context.
- **pRspBuf** – Pointer to response Buffer.
- **pRspBufLen** – Pointer to length of the response Buffer

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

10.15.4.217 Function `sems_lite_get_FIPS_EC_parameter_type`

- Defined in [File sems_lite_api.h](#)

10.15.4.217.1 Function Documentation

`sss_status_t sems_lite_get_FIPS_EC_parameter_type(sems_lite_agent_ctx_t *pContext, uint8_t *pParamType)`

Get Configured EC domain parameter type.

This API will return Configured EC domain parameter type

Parameters

- **pContext** – Pointer to sems lite agent context.
- **pParamType** – Pointer to parameter type.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

10.15.4.218 Function `sems_lite_get_FIPS_info`

- Defined in [File sems_lite_api.h](#)

10.15.4.218.1 Function Documentation

`sss_status_t sems_lite_get_FIPS_info(sems_lite_agent_ctx_t *pContext, uint8_t *pFIPSInfo)`

Get Configured FIPS Information.

This API will return Configured FIPS Information

Parameters

- `pContext` – Pointer to sems lite agent context.
- `pFIPSInfo` – Pointer to FIPS Info.

Return values

- `kStatus_SSS_Success` – The operation has completed successfully.
- `kStatus_SSS_Fail` – The operation has failed.

Returns

Status of the operation

10.15.4.219 Function `sems_lite_get_Publickey`

- Defined in [File sems_lite_api.h](#)

10.15.4.219.1 Function Documentation

`sss_status_t sems_lite_get_Publickey(sems_lite_agent_ctx_t *pContext, uint8_t *pRspBuf, size_t *pRspBufLen)`

Read Public Key.

This API will read root certificates public key of the device.

Parameters

- `pContext` – Pointer to sems lite agent context.
- `pRspBuf` – Pointer to response Buffer.
- `pRspBufLen` – Pointer to length of the response Buffer.

Return values

- `kStatus_SSS_Success` – The operation has completed successfully.
- `kStatus_SSS_Fail` – The operation has failed.

Returns

Status of the operation

10.15.4.220 Function `sems_lite_get_SEAppInfo`

- Defined in [File sems_lite_api.h](#)

10.15.4.220.1 Function Documentation

`sss_status_t sems_lite_get_SEAppInfo(sems_lite_agent_ctx_t *pContext, const uint8_t *searchAID, uint8_t searchAIDLen, sems_lite_SEAppInfoList_t *pAppInfo, size_t *pAppInfoLen)`

Low level API to get App INFO from the SE according to format mentioned in - Table 11-36: GlobalPlatform Registry Data (TLV), GPCardSpc_v2.2.pdf.

This API will read the currently present Applications ELF/ELM AIDs and versions as well as the present instances from the SE.

Parameters

- **pContext** – Pointer to sems lite agent context.
- **searchAID** – **[in]** The search aid
- **searchAidLen** – **[in]** The search aid length
- **pAppInfo** – Parsed structures
- **pAppInfoLen** – **[inout]** Length of parsed structures.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

10.15.4.221 Function `sems_lite_get_SEAppInfoRAW`

- Defined in [File sems_lite_api.h](#)

10.15.4.221.1 Function Documentation

`sss_status_t sems_lite_get_SEAppInfoRAW(sems_lite_agent_ctx_t *pContext, const uint8_t *searchAID, uint8_t searchAidLen, uint8_t *pRspBuf, size_t *pRspBufLen)`

Low level API to get Raw App INFO from the SE.

This API will read the currently present Applications ELF/ELM AIDs and versions as well as the present instances from the SE.

Parameters

- **pContext** – Pointer to sems lite agent context.
- **pRspBuf** – **[out]** Pointer to response Buffer.
- **searchAID** – **[in]** The search aid
- **searchAidLen** – **[in]** The search aid length
- **pRspBufLen** – **[inout]** Pointer to length of the response Buffer.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

10.15.4.222 Function `sems_lite_get_SEPkgInfo`

- Defined in [File sems_lite_api.h](#)

10.15.4.222.1 Function Documentation

`sss_status_t sems_lite_get_SEPkgInfo(sems_lite_agent_ctx_t *pContext, const uint8_t *searchAID, uint8_t searchAidLen, sems_lite_SEAppInfoList_t *pAppInfo, size_t *pAppInfoLen)`

Low level API to get PKG INFO from the SE according to format mentioned in - Table 11-36: GlobalPlatform Registry Data (TLV), GPCardSpc_v2.2.pdf.

This API will read the currently present Applications ELF/ELM AIDs and versions as well as the present instances from the SE.

Parameters

- **pContext** – Pointer to sems lite agent context.
- **searchAID** – [in] The search aid
- **searchAidLen** – [in] The search aid length
- **pAppInfo** – Parsed structures
- **pAppInfoLen** – [inout] Length of parsed structures.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

10.15.4.223 Function `sems_lite_get_SEPkglInfoRAW`

- Defined in [File sems_lite_api.h](#)

10.15.4.223.1 Function Documentation

`sss_status_t sems_lite_get_SEPkglInfoRAW(sems_lite_agent_ctx_t *pContext, const uint8_t *searchAID, uint8_t searchAidLen, uint8_t *pRspBuf, size_t *pRspBufLen)`

Low level API to get RAW PKG INFO from the SE.

This API will read the currently present Applications ELF/ELM AIDs and versions as well as the present instances from the SE.

Parameters

- **pContext** – Pointer to sems lite agent context.
- **searchAID** – [in] The search aid
- **searchAidLen** – [in] The search aid length
- **pRspBuf** – [out] Pointer to response Buffer.
- **pRspBufLen** – [inout] Pointer to length of the response Buffer.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

10.15.4.224 Function `sems_lite_get_SignatureofLastScript`

- Defined in [File sems_lite_api.h](#)

10.15.4.224.1 Function Documentation

`sss_status_t sems_lite_get_SignatureofLastScript(sems_lite_agent_ctx_t *pContext, uint8_t *pRspBuf, size_t *pRspBufLen)`

Get the signature of last executed script.

This API will called in case there is tear down

Parameters

- **pContext** – Pointer to sems lite agent context.
- **pRspBuf** – Pointer to response Buffer.
- **pRspBufLen** – Pointer to length of the response Buffer

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

10.15.4.225 Function `sems_lite_get_UUID`

- Defined in [File `sems_lite_api.h`](#)

10.15.4.225.1 Function Documentation

`sss_status_t sems_lite_get_UUID(sss_lite_agent_ctxt_t *pContext, uint8_t *pRspBuf, size_t *pRspBufLen)`

Retrieve UUID from SE.

This API read UUID of the SE.

Parameters

- **pContext** – Pointer to sems lite agent context.
- **pRspBuf** – Pointer to response Buffer.
- **pRspBufLen** – Pointer to length of the response Buffer.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

10.15.4.226 Function `sss_aead_context_free`

- Defined in [File `fsl_sss_api.h`](#)

10.15.4.226.1 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.

10.15.4.227 Function `sss_aead_context_init`

- Defined in [File `fsl_sss_api.h`](#)

10.15.4.227.1 Function Documentation

`sss_status_t sss_aead_context_init(sss_aead_t *context, sss_session_t *session, sss_object_t *keyObject, sss_algorithm_t algorithm, sss_mode_t mode)`

AEAD context init. The function initializes aead context with initial values.

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 values

- **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.

Returns

Status of the operation

10.15.4.228 Function sss_aead_finish

- Defined in [File fsl_sss_api.h](#)

10.15.4.228.1 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).

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.
- **destLen** – **[inout]** 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 values

- **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.

Returns

Status of the operation

10.15.4.229 Function sss_aead_init

- Defined in [File fsl_sss_api.h](#)

10.15.4.229.1 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.

Parameters

- context** – Pointer to aead crypto context.
- nonce** – The operation nonce or IV. When using internal IV algorithms (only encrypt) for SE051, iv buffer will be filled with generated Initialization Vector.
- nonceLen** – The length of nonce in bytes. For AES-GCM it must be ≥ 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 values

- kStatus_SSS_Success** – The operation has completed successfully.
- kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

10.15.4.230 Function sss_aead_one_go

- Defined in [File fsl_sss_api.h](#)

10.15.4.230.1 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.

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. When using internal IV algorithms (only encrypt) for SE051, iv buffer will be filled with generated Initialization Vector.
- **nonceLen** – The length of nonce in bytes. For AES-GCM it must be ≥ 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 (Default value is 16 bytes). 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 values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

10.15.4.231 Function `sss_aead_update`

- Defined in [File fsl_sss_api.h](#)

10.15.4.231.1 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.

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.
- **destLen** – **[inout]** Length of the output data in bytes. Buffer length on entry, reflects actual output size on return.

Return values

- **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.

Returns

Status of the operation

10.15.4.232 Function `sss_aead_update_aad`

- Defined in [File fsl_sss_api.h](#)

10.15.4.232.1 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.

Parameters

- `context` – Pointer to aead crypto context
- `aadData` – Input buffer containing the chunk of AAD
- `aadDataLen` – Length of the AAD data in bytes.

Return values

- `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.

Returns

Status of the operation

10.15.4.233 Function `sss_asymmetric_context_free`

- Defined in [File fsl_sss_api.h](#)

10.15.4.233.1 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.

10.15.4.234 Function `sss_asymmetric_context_init`

- Defined in [File fsl_sss_api.h](#)

10.15.4.234.1 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.

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 values

- **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.

Returns

Status of the operation

10.15.4.235 Function `sss_asymmetric_decrypt`

- Defined in [File fsl_sss_api.h](#)

10.15.4.235.1 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.

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 values

- **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.

Returns

Status of the operation

10.15.4.236 Function `sss_asymmetric_encrypt`

- Defined in [File fsl_sss_api.h](#)

10.15.4.236.1 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.

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 values

- **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.

Returns

Status of the operation

10.15.4.237 Function `sss_asymmetric_sign_digest`

- Defined in [File fsl_sss_api.h](#)

10.15.4.237.1 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.

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 values

- **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.

Returns

Status of the operation

10.15.4.238 Function `sss_asymmetric_verify_digest`

- Defined in [File fsl_sss_api.h](#)

10.15.4.238.1 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.

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 values

- **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.

Returns

Status of the operation

10.15.4.239 Function `sss_cipher_crypt_ctr`

- Defined in [File fsl_sss_api.h](#)

10.15.4.239.1 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.

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.
- **initialCounter** – **[inout]** Input counter (Always 16 bytes) (updates on return). When using internal IV algorithms (only encrypt) for SE051, initialCounter buffer will be filled with generated Initial counter.
- **lastEncryptedCounter** – **[out]** Output cipher of last counter, for chained CTR calls. NULL can be passed if chained calls are not used.
- **szLeft** – **[out]** Output number of bytes in left unused in lastEncryptedCounter block. NULL can be passed if chained calls are not used.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

10.15.4.240 Function `sss_cipher_finish`

- Defined in [File fsl_sss_api.h](#)

10.15.4.240.1 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.

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.
- `destLen` – **[inout]** Length of output data in bytes. Buffer length on entry, reflects actual output size on return.

Return values

- `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.

Returns

Status of the operation

10.15.4.241 Function sss_cipher_init

- Defined in [File fsl_sss_api.h](#)

10.15.4.241.1 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.

Parameters

- `context` – Pointer to symmetric crypto context.
- `iv` – Buffer containing the symmetric operation Initialization Vector. When using internal IV algorithms (only encrypt) for SE051, iv buffer will be filled with generated Initialization Vector.
- `ivLen` – Length of the Initialization Vector in bytes.

Return values

- `kStatus_SSS_Success` – The operation has completed successfully.
- `kStatus_SSS_Fail` – The operation has failed.

Returns

Status of the operation

10.15.4.242 Function sss_cipher_one_go

- Defined in [File fsl_sss_api.h](#)

10.15.4.242.1 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.

Parameters

- **context** – Pointer to symmetric crypto context.
- **iv** – Buffer containing the symmetric operation Initialization Vector. When using internal IV algorithms (only encrypt) for SE051, iv buffer will be filled with generated Initialization Vector.
- **ivLen** – Length of the Initialization Vector in bytes.
- **srcData** – Buffer containing the input data (block aligned).
- **destData** – Buffer containing the output data.
- **dataLen** – Size of input and output data buffer in bytes.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

10.15.4.243 Function sss_cipher_one_go_v2

- Defined in [File fsl_sss_api.h](#)

10.15.4.243.1 Function Documentation

`sss_status_t sss_cipher_one_go_v2(sss_symmetric_t *context, uint8_t *iv, size_t ivLen, const uint8_t *srcData, const size_t srcLen, 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.

Parameters

- **context** – Pointer to symmetric crypto context.
- **iv** – Buffer containing the symmetric operation Initialization Vector. When using internal IV algorithms (only encrypt) for SE051, iv buffer will be filled with generated Initialization Vector.
- **ivLen** – Length of the Initialization Vector in bytes.
- **srcData** – Buffer containing the input data (block aligned).
- **srcLen** – Length of buffer srcData.
- **destData** – Buffer containing the output data.
- **dataLen** – Pointer to Size of buffer destData in bytes.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

10.15.4.244 Function `sss_cipher_type_sz`

- Defined in [File fsl_sss_api.h](#)

10.15.4.244.1 Function Documentation

```
const char *sss_cipher_type_sz(sss\_cipher\_type\_t cipher_type)
```

Returns string error code for [sss_cipher_type_t](#).

Parameters

`cipher_type` – [in] See [sss_cipher_type_t](#)

Returns

String conversion of `cipher_type` to String.

10.15.4.245 Function `sss_cipher_update`

- Defined in [File fsl_sss_api.h](#)

10.15.4.245.1 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\(\)](#).

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.
- `destLen` – [inout] Length of the output data in bytes. Buffer length on entry, reflects actual output size on return.

Return values

- `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.

Returns

Status of the operation

10.15.4.246 Function `sss_derive_key_context_free`

- Defined in [File fsl_sss_api.h](#)

10.15.4.246.1 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.

10.15.4.247 Function sss_derive_key_context_init

- Defined in [File fsl_sss_api.h](#)

10.15.4.247.1 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.

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 values

- 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.

Returns

Status of the operation

10.15.4.248 Function sss_derive_key_dh

- Defined in [File fsl_sss_api.h](#)

10.15.4.248.1 Function Documentation

sss_status_t sss_derive_key_dh(sss_derive_key_t *context, sss_object_t *otherPartyKeyObject, sss_object_t *derived KeyObject)

Asymmetric key derivation Diffie-Hellmann The function cryptographically derives a key from another key. For example Diffie-Hellmann.

Parameters

- context** – Pointer to derive key context.
- otherPartyKeyObject** – Public key of the other party in the Diffie-Hellmann algorithm
- derivedKeyObject** – [inout] Reference to a derived key

Return values

- 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.

Returns

Status of the operation

10.15.4.249 Function sss_derive_key_go

- Defined in [File fsl_sss_api.h](#)

10.15.4.249.1 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.

Deprecated:

Please use [sss_derive_key_one_go](#) instead

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.
- derivedKeyObject** – **[inout]** Reference to a derived key
- deriveDataLen** – Requested length of output
- hkdfOutput** – Output buffer containing key derivation output
- hkdfOutputLen** – Output containing length of hkdfOutput

Return values

- 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.

Returns

Status of the operation

10.15.4.250 Function sss_derive_key_one_go

- Defined in [File fsl_sss_api.h](#)

10.15.4.250.1 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.

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.
- **derivedKeyObject** – **[inout]** Reference to a derived key
- **deriveDataLen** – **[in]** Expected length of derived key.

Return values

- **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.

Returns

Status of the operation

10.15.4.251 Function `sss_derive_key_sobj_one_go`

- Defined in [File fsl_sss_api.h](#)

10.15.4.251.1 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.

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.
- **info** – **[in]** Input data buffer, typically with some fixed info.
- **infoLen** – **[in]** Length of info buffer in bytes.
- **derivedKeyObject** – Reference to a derived key
- **deriveDataLen** – **[in]** The derive data length

Return values

- **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.

Returns

Status of the operation

10.15.4.252 Function `sss_digest_context_free`

- Defined in [File fsl_sss_api.h](#)

10.15.4.252.1 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.

10.15.4.253 Function sss_digest_context_init

- Defined in [File fsl_sss_api.h](#)

10.15.4.253.1 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.

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 values

- `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.

Returns

Status of the operation

10.15.4.254 Function sss_digest_finish

- Defined in [File fsl_sss_api.h](#)

10.15.4.254.1 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.

Parameters

- `context` – Pointer to digest context.
- `digest` – Output message digest
- `digestLen` – Message digest byte length

Return values

- `kStatus_SSS_Success` – The operation has completed successfully.
- `kStatus_SSS_Fail` – The operation has failed.

Returns

Status of the operation

10.15.4.255 Function sss_digest_init

- Defined in [File fsl_sss_api.h](#)

10.15.4.255.1 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.

Parameters

`context` – Pointer to digest context.

Return values

- `kStatus_SSS_Success` – The operation has completed successfully.
- `kStatus_SSS_Fail` – The operation has failed.

Returns

Status of the operation

10.15.4.256 Function sss_digest_one_go

- Defined in [File fsl_sss_api.h](#)

10.15.4.256.1 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.

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 values

- `kStatus_SSS_Success` – The operation has completed successfully.
- `kStatus_SSS_Fail` – The operation has failed.

Returns

Status of the operation

10.15.4.257 Function sss_digest_update

- Defined in [File fsl_sss_api.h](#)

10.15.4.257.1 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.

Parameters

- `context` – Pointer to digest context.
- `message` – Buffer with a message chunk.
- `messageLen` – Length of the input buffer in bytes.

Return values

- `kStatus_SSS_Success` – The operation has completed successfully.
- `kStatus_SSS_Fail` – The operation has failed.

Returns

Status of the operation

10.15.4.258 Function `sss_key_object_allocate_handle`

- Defined in [File fsl_sss_api.h](#)

10.15.4.258.1 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.

Parameters

- `keyObject` – [inout] 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 = Persistant Key (Default) or Transient Key. See [sss_key_object_mode_t](#)

Returns

Status of object allocation.

10.15.4.259 Function `sss_key_object_free`

- Defined in [File fsl_sss_api.h](#)

10.15.4.259.1 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.

10.15.4.260 Function sss_key_object_get_access

- Defined in [File fsl_sss_api.h](#)

10.15.4.260.1 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

Parameters

- **keyObject** – Object
- **access** – What is permitted

Returns

10.15.4.261 Function sss_key_object_get_handle

- Defined in [File fsl_sss_api.h](#)

10.15.4.261.1 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 @ref sss_key_object_allocate_handle.

After calling this API, Ideally keyObject should become equivlant to as set after the calling of @ref sss_key_object_allocate_handle api.

Parameters

- **keyObject** – The key object
- **keyId** – [in] The key identifier

Returns

The sss status.

10.15.4.262 Function sss_key_object_get_purpose

- Defined in [File fsl_sss_api.h](#)

10.15.4.262.1 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

Parameters

- **keyObject** – Object to be checked
- **purpose** – Know what is permitted.

Returns**10.15.4.263 Function sss_key_object_get_user**

- Defined in [File fsl_sss_api.h](#)

10.15.4.263.1 Function Documentation

`sss_status_t sss_key_object_get_user(sss_object_t *keyObject, uint32_t *user)`

get attributes

10.15.4.264 Function sss_key_object_init

- Defined in [File fsl_sss_api.h](#)

10.15.4.264.1 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.

Parameters

- `keyObject` –
- `keyStore` –

Return values

- `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.

Returns

Status of the operation

10.15.4.265 Function sss_key_object_set_access

- Defined in [File fsl_sss_api.h](#)

10.15.4.265.1 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.

10.15.4.266 Function sss_key_object_set_eccgfp_group

- Defined in [File fsl_sss_api.h](#)

10.15.4.266.1 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)

10.15.4.267 Function sss_key_object_set_purpose

- Defined in [File fsl_sss_api.h](#)

10.15.4.267.1 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.

10.15.4.268 Function sss_key_object_set_user

- Defined in [File fsl_sss_api.h](#)

10.15.4.268.1 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.

10.15.4.269 Function sss_key_store_allocate

- Defined in [File fsl_sss_api.h](#)

10.15.4.269.1 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 exists, new empty key store is created and initialized. Key store context structure is updated with actual information.

Parameters

- **keyStore** – **[out]** 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.

10.15.4.270 Function `sss_key_store_context_free`

- Defined in [File fsl_sss_api.h](#)

10.15.4.270.1 Function Documentation

`void sss_key_store_context_free(sss_key_store_t *keyStore)`

Destructor for the key store context.

10.15.4.271 Function `sss_key_store_context_init`

- Defined in [File fsl_sss_api.h](#)

10.15.4.271.1 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

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

10.15.4.272 Function `sss_key_store_erase_key`

- Defined in [File fsl_sss_api.h](#)

10.15.4.272.1 Function Documentation

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

Delete / destroy allocated keyObject .

Parameters

- **keyStore** – The key store
- **keyObject** – The key object to be deleted

Returns

The sss status.

10.15.4.273 Function sss_key_store_freeze_key

- Defined in [File fsl_sss_api.h](#)

10.15.4.273.1 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.

Parameters

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

Returns

The sss status.

10.15.4.274 Function sss_key_store_generate_key

- Defined in [File fsl_sss_api.h](#)

10.15.4.274.1 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.

10.15.4.275 Function sss_key_store_get_key

- Defined in [File fsl_sss_api.h](#)

10.15.4.275.1 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)

10.15.4.276 Function sss_key_store_load

- Defined in [File fsl_sss_api.h](#)

10.15.4.276.1 Function Documentation

`sss_status_t sss_key_store_load(sss_key_store_t *keyStore)`

Load from persistent memory to cached objects.

10.15.4.277 Function sss_key_store_open_key

- Defined in [File fsl_sss_api.h](#)

10.15.4.277.1 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

Parameters

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

Returns

The sss status.

10.15.4.278 Function sss_key_store_save

- Defined in [File fsl_sss_api.h](#)

10.15.4.278.1 Function Documentation

`sss_status_t sss_key_store_save(sss_key_store_t *keyStore)`

Save all cached persistent objects to persistent memory.

10.15.4.279 Function sss_key_store_set_key

- Defined in [File fsl_sss_api.h](#)

10.15.4.279.1 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.

Parameters

- `keyStore` – Key store context
- `keyObject` – Reference to a key and its 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

Returns

10.15.4.280 Function sss_mac_context_free

- Defined in [File fsl_sss_api.h](#)

10.15.4.280.1 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.

10.15.4.281 Function `sss_mac_context_init`

- Defined in [File fsl_sss_api.h](#)

10.15.4.281.1 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.

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 values

- **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.

Returns

Status of the operation

10.15.4.282 Function `sss_mac_finish`

- Defined in [File fsl_sss_api.h](#)

10.15.4.282.1 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.

Parameters

- **context** – Pointer to mac context.
- **mac** – Output message MAC
- **macLen** – Computed MAC byte length

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

10.15.4.283 Function sss_mac_init

- Defined in [File fsl_sss_api.h](#)

10.15.4.283.1 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.

Parameters

`context` – Pointer to mac context.

Return values

- `kStatus_SSS_Success` – The operation has completed successfully.
- `kStatus_SSS_Fail` – The operation has failed.

Returns

Status of the operation

10.15.4.284 Function sss_mac_one_go

- Defined in [File fsl_sss_api.h](#)

10.15.4.284.1 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.

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 values

- `kStatus_SSS_Success` – The operation has completed successfully.
- `kStatus_SSS_Fail` – The operation has failed.

Returns

Status of the operation

10.15.4.285 Function sss_mac_update

- Defined in [File fsl_sss_api.h](#)

10.15.4.285.1 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.

Parameters

- **context** – Pointer to mac context.
- **message** – Buffer with a message chunk.
- **messageLen** – Length of the input buffer in bytes.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

10.15.4.286 Function `sss_mbedtls_associate_ecdhctx`

- Defined in [File sss_mbedtls.h](#)

10.15.4.286.1 Function Documentation

```
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 index.

Parameters

- **handshake** – **[inout]** Pointer to the mbedtls_ssl_handshake_params which will be associated with data corresponding to the key index
- **pkeyObject** – **[in]** The object that we are going to be use.
- **hostKs** – **[in]** Keystore to host for session key.

Returns

0 if successful, or 1 if unsuccessful

10.15.4.287 Function `sss_mbedtls_associate_keypair`

- Defined in [File sss_mbedtls.h](#)

10.15.4.287.1 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.

Parameters

- **pkey** – **[out]** Pointer to the mbedtls_pk_context which will be associated with data corresponding to the key_index
- **pkeyObject** – **[in]** The object that we are going to be use.

Returns

0 if successful, or 1 if unsuccessful

10.15.4.288 Function `sss_mbedtls_associate_pubkey`

- Defined in [File sss_mbedtls.h](#)

10.15.4.288.1 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.

Parameters

- **pkey** – **[out]** Pointer to the mbedtls_pk_context which will be associated with data corresponding to the key index
- **pkeyObject** – **[in]** The object that we are going to be use.

Returns

0 if successful, or 1 if unsuccessful

10.15.4.289 Function sss_rng_context_free

- Defined in [File fsl_sss_api.h](#)

10.15.4.289.1 Function Documentation

`sss_status_t sss_rng_context_free(sss_rng_context_t *context)`

free random generator context.

Parameters

context – generator context.

Returns

status

10.15.4.290 Function sss_rng_context_init

- Defined in [File fsl_sss_api.h](#)

10.15.4.290.1 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);%%CRLF  
SP%%Now: Parameters are swapped  
sss_status_t sss_rng_context_init(  
    sss_rng_context_t *context, sss_session_t *session);
```

Parameters

- **session** – Session context.
- **context** – random generator context.

Returns

status

10.15.4.291 Function `sss_rng_get_random`

- Defined in [File fsl_sss_api.h](#)

10.15.4.291.1 Function Documentation

`sss_status_t sss_rng_get_random(sss_rng_context_t *context, uint8_t *random_data, size_t dataLen)`

Generate random number.

Parameters

- `context` – random generator context.
- `random_data` – buffer to hold random data.
- `dataLen` – required random number length

Returns

`status`

10.15.4.292 Function `sss_se05x_aead_context_free`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.292.1 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.

10.15.4.293 Function `sss_se05x_aead_context_init`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.293.1 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.

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 values

- `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.

Returns

Status of the operation

10.15.4.294 Function sss_se05x_aead_finish

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.294.1 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 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).

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.
- destLen** – **[inout]** 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 values

- 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.

Returns

Status of the operation

10.15.4.295 Function sss_se05x_aead_init

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.295.1 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.

Parameters

- **context** – Pointer to aead crypto context.
- **nonce** – The operation nonce or IV. When using internal IV algorithms (only encrypt) for SE051, iv buffer will be filled with generated Initialization Vector.
- **nonceLen** – The length of nonce in bytes. For AES-GCM it must be ≥ 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 values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

10.15.4.296 Function `sss_se05x_aead_one_go`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.296.1 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.

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. When using internal IV algorithms (only encrypt) for SE051, iv buffer will be filled with generated Initialization Vector.
- **nonceLen** – The length of nonce in bytes. For AES-GCM it must be ≥ 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 (Default value is 16 bytes). 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 values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

10.15.4.297 Function sss_se05x_aead_update

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.297.1 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.

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.
- destLen** – **[inout]** Length of the output data in bytes. Buffer length on entry, reflects actual output size on return.

Return values

- 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.

Returns

Status of the operation

10.15.4.298 Function sss_se05x_aead_update_aad

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.298.1 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.

Parameters

- context** – Pointer to aead crypto context
- aadData** – Input buffer containing the chunk of AAD
- aadDataLen** – Length of the AAD data in bytes.

Return values

- 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.

Returns

Status of the operation

10.15.4.299 Function sss_se05x_asymmetric_context_free

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.299.1 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.

10.15.4.300 Function sss_se05x_asymmetric_context_init

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.300.1 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.

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 values

- `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.

Returns

Status of the operation

10.15.4.301 Function sss_se05x_asymmetric_decrypt

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.301.1 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.

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 values

- **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.

Returns

Status of the operation

10.15.4.302 Function `sss_se05x_asymmetric_encrypt`

- Defined in [File fsl_sss_se05x/apis.h](#)

10.15.4.302.1 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.

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 values

- **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.

Returns

Status of the operation

10.15.4.303 Function `sss_se05x_asymmetric_sign`

- Defined in [File fsl_sss_se05x/types.h](#)

10.15.4.303.1 Function Documentation

`sss_status_t sss_se05x_asymmetric_sign(sss_se05x_asymmetric_t *context, const 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

10.15.4.304 Function `sss_se05x_asymmetric_sign_digest`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.304.1 Function Documentation

`sss_status_t sss_se05x_asymmetric_sign_digest(sss_se05x_asymmetric_t *context, const uint8_t *digest, size_t digestLen, uint8_t *signature, size_t *signatureLen)`

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

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 values

- `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.

Returns

Status of the operation

10.15.4.305 Function `sss_se05x_asymmetric_verify`

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.4.305.1 Function Documentation

`sss_status_t sss_se05x_asymmetric_verify(sss_se05x_asymmetric_t *context, const uint8_t *srcData, size_t srcLen, const uint8_t *signature, size_t signatureLen)`

Similar to [sss_se05x_asymmetric_verify_digest](#), but hashing/digest done by SE

10.15.4.306 Function `sss_se05x_asymmetric_verify_digest`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.306.1 Function Documentation

`sss_status_t sss_se05x_asymmetric_verify_digest(sss_se05x_asymmetric_t *context, const uint8_t *digest, size_t digestLen, const uint8_t *signature, size_t signatureLen)`

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

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 values

- **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.

Returns

Status of the operation

10.15.4.307 Function `sss_se05x_cipher_crypt_ctr`

- Defined in [File fsl_sss_se05x/apis.h](#)

10.15.4.307.1 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.

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.
- **initialCounter** – **[inout]** Input counter (Always 16 bytes) (updates on return). When using internal IV algorithms (only encrypt) for SE051, initialCounter buffer will be filled with generated Initial counter.
- **lastEncryptedCounter** – **[out]** Output cipher of last counter, for chained CTR calls. NULL can be passed if chained calls are not used.
- **szLeft** – **[out]** Output number of bytes in left unused in lastEncryptedCounter block. NULL can be passed if chained calls are not used.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

10.15.4.308 Function `sss_se05x_cipher_finish`

- Defined in [File fsl_sss_se05x/apis.h](#)

10.15.4.308.1 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.

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.
- **destLen** – **[inout]** Length of output data in bytes. Buffer length on entry, reflects actual output size on return.

Return values

- **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.

Returns

Status of the operation

10.15.4.309 Function sss_se05x_cipher_init

- Defined in [File fsl_sss_se05x/apis.h](#)

10.15.4.309.1 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.

Parameters

- **context** – Pointer to symmetric crypto context.
- **iv** – Buffer containing the symmetric operation Initialization Vector. When using internal IV algorithms (only encrypt) for SE051, iv buffer will be filled with generated Initialization Vector.
- **ivLen** – Length of the Initialization Vector in bytes.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

10.15.4.310 Function sss_se05x_cipher_one_go

- Defined in [File fsl_sss_se05x/apis.h](#)

10.15.4.310.1 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.

Parameters

- **context** – Pointer to symmetric crypto context.
- **iv** – Buffer containing the symmetric operation Initialization Vector. When using internal IV algorithms (only encrypt) for SE051, iv buffer will be filled with generated Initialization Vector.
- **ivLen** – Length of the Initialization Vector in bytes.
- **srcData** – Buffer containing the input data (block aligned).
- **destData** – Buffer containing the output data.
- **dataLen** – Size of input and output data buffer in bytes.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

10.15.4.311 Function sss_se05x_cipher_one_go_v2

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.311.1 Function Documentation

`sss_status_t sss_se05x_cipher_one_go_v2(sss_se05x_symmetric_t *context, uint8_t *iv, size_t ivLen, const uint8_t *srcData, const size_t srcLen, uint8_t *destData, size_t *pDataLen)`

Symmetric cipher in one blocking function call. The function blocks current thread until the operation completes or an error occurs.

Parameters

- **context** – Pointer to symmetric crypto context.
- **iv** – Buffer containing the symmetric operation Initialization Vector. When using internal IV algorithms (only encrypt) for SE051, iv buffer will be filled with generated Initialization Vector.
- **ivLen** – Length of the Initialization Vector in bytes.
- **srcData** – Buffer containing the input data (block aligned).
- **srcLen** – Length of buffer srcData.
- **destData** – Buffer containing the output data.
- **pDataLen** – Pointer to Size of buffer destData in bytes.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

10.15.4.312 Function `sss_se05x_cipher_update`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.312.1 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\(\)](#).

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.
- `destLen` – **[inout]** Length of the output data in bytes. Buffer length on entry, reflects actual output size on return.

Return values

- `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.

Returns

Status of the operation

10.15.4.313 Function `sss_se05x_derive_key_context_free`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.313.1 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.

10.15.4.314 Function `sss_se05x_derive_key_context_init`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.314.1 Function Documentation

`sss_status_t sss_se05x_derive_key_context_init(sss_se05x_derive_key_t *context, sss_se05x_session_t *session, 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.

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 values

- **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.

Returns

Status of the operation

10.15.4.315 Function `sss_se05x_derive_key_dh`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.315.1 Function Documentation

`sss_status_t sss_se05x_derive_key_dh(sss_se05x_derive_key_t *context, sss_se05x_object_t *otherPartyKeyObject, sss_se05x_object_t *derivedKeyObject)`

Asymmetric key derivation Diffie-Hellmann The function cryptographically derives a key from another key. For example Diffie-Hellmann.

Parameters

- **context** – Pointer to derive key context.
- **otherPartyKeyObject** – Public key of the other party in the Diffie-Hellmann algorithm
- **derivedKeyObject** – **[inout]** Reference to a derived key

Return values

- **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.

Returns

Status of the operation

10.15.4.316 Function `sss_se05x_derive_key_go`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.316.1 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.

Deprecated:

Please use [sss_derive_key_one_go](#) instead

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.
- **derivedKeyObject** – **[inout]** Reference to a derived key
- **deriveDataLen** – Requested length of output
- **hkdfOutput** – Output buffer containing key derivation output
- **hkdfOutputLen** – Output containing length of hkdfOutput

Return values

- **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.

Returns

Status of the operation

10.15.4.317 Function `sss_se05x_derive_key_one_go`

- Defined in [File fsl_sss_se05x/apis.h](#)

10.15.4.317.1 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.

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.
- **derivedKeyObject** – **[inout]** Reference to a derived key
- **deriveDataLen** – **[in]** Expected length of derived key.

Return values

- **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.

Returns

Status of the operation

10.15.4.318 Function sss_se05x_derive_key_sobj_one_go

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.318.1 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.

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.
- `info` – [in] Input data buffer, typically with some fixed info.
- `infoLen` – [in] Length of info buffer in bytes.
- `derivedKeyObject` – Reference to a derived key
- `deriveDataLen` – [in] The derive data length

Return values

- `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.

Returns

Status of the operation

10.15.4.319 Function sss_se05x_digest_context_free

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.319.1 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.

10.15.4.320 Function sss_se05x_digest_context_init

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.320.1 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.

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 values

- `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.

Returns

Status of the operation

10.15.4.321 Function sss_se05x_digest_finish

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.321.1 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.

Parameters

- `context` – Pointer to digest context.
- `digest` – Output message digest
- `digestLen` – Message digest byte length

Return values

- `kStatus_SSS_Success` – The operation has completed successfully.
- `kStatus_SSS_Fail` – The operation has failed.

Returns

Status of the operation

10.15.4.322 Function sss_se05x_digest_init

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.322.1 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.

Parameters

`context` – Pointer to digest context.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

10.15.4.323 Function `sss_se05x_digest_one_go`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.323.1 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.

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 values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

10.15.4.324 Function `sss_se05x_digest_update`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.324.1 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.

Parameters

- **context** – Pointer to digest context.
- **message** – Buffer with a message chunk.
- **messageLen** – Length of the input buffer in bytes.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

10.15.4.325 Function sss_se05x_key_object_allocate_handle

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.325.1 Function Documentation

`sss_status_t sss_se05x_key_object_allocate_handle(sss_se05x_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.

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.

Parameters

- keyObject** – **[inout]** 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 = Persistant Key (Default) or Transient Key. See [sss_key_object_mode_t](#)

Returns

Status of object allocation.

10.15.4.326 Function sss_se05x_key_object_free

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.326.1 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.

10.15.4.327 Function sss_se05x_key_object_get_access

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.327.1 Function Documentation

`sss_status_t sss_se05x_key_object_get_access(sss_se05x_object_t *keyObject, uint32_t *access)`

Not Available for SE05X

10.15.4.328 Function `sss_se05x_key_object_get_handle`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.328.1 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 @ref `sss_key_object_allocate_handle`.

After calling this API, Ideally `keyObject` should become equivlant to as set after the calling of @ref `sss_key_object_allocate_handle` api.

On SE05X, this API uses [Se05x_API_ReadType](#) and fetches parameters of the API.

Parameters

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

Returns

The `sss status`.

10.15.4.329 Function `sss_se05x_key_object_get_purpose`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.329.1 Function Documentation

`sss_status_t sss_se05x_key_object_get_purpose(sss_se05x_object_t *keyObject, sss_mode_t *purpose)`

Not Available for SE05X

10.15.4.330 Function `sss_se05x_key_object_get_user`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.330.1 Function Documentation

`sss_status_t sss_se05x_key_object_get_user(sss_se05x_object_t *keyObject, uint32_t *user)`

Not Available for SE05X

10.15.4.331 Function `sss_se05x_key_object_init`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.331.1 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.

Parameters

- `keyObject` –
- `keyStore` –

Return values

- `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.

Returns

Status of the operation

10.15.4.332 Function `sss_se05x_key_object_set_access`

- Defined in [File fsl_sss_se05x/apis.h](#)

10.15.4.332.1 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

10.15.4.333 Function `sss_se05x_key_object_set_eccgfp_group`

- Defined in [File fsl_sss_se05x/apis.h](#)

10.15.4.333.1 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

10.15.4.334 Function `sss_se05x_key_object_set_purpose`

- Defined in [File fsl_sss_se05x/apis.h](#)

10.15.4.334.1 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.

10.15.4.335 Function sss_se05x_key_object_set_user

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.335.1 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

10.15.4.336 Function sss_se05x_key_store_allocate

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.336.1 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 exists, 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

- **keyStore** – **[out]** 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.

10.15.4.337 Function sss_se05x_key_store_context_free

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.337.1 Function Documentation

`void sss_se05x_key_store_context_free(sss_se05x_key_store_t *keyStore)`

Destructor for the key store context.

10.15.4.338 Function sss_se05x_key_store_context_init

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.338.1 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

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

10.15.4.339 Function sss_se05x_key_store_create_curve

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.339.1 Function Documentation

`sss_status_t sss_se05x_key_store_create_curve(Se05xSession_t *pSession, uint32_t curve_id)`

Creates a key store entry for specified curve

10.15.4.340 Function sss_se05x_key_store_erase_key

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.340.1 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 keyObject .

Parameters

- **keyStore** – The key store
- **keyObject** – The key object to be deleted

Returns

The sss status.

10.15.4.341 Function sss_se05x_key_store_export_key

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.341.1 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.

10.15.4.342 Function sss_se05x_key_store_freeze_key

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.342.1 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

10.15.4.343 Function `sss_se05x_key_store_generate_key`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.343.1 Function Documentation

`sss_status_t sss_se05x_key_store_generate_key(sss_se05x_key_store_t *keyStore, sss_se05x_object_t *keyObject, size_t keyBitLen, void *options)`

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

10.15.4.344 Function `sss_se05x_key_store_get_key`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.344.1 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)

10.15.4.345 Function `sss_se05x_key_store_get_key_attst`

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.4.345.1 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

10.15.4.346 Function `sss_se05x_key_store_import_key`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.346.1 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.

10.15.4.347 Function `sss_se05x_key_store_load`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.347.1 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.

10.15.4.348 Function `sss_se05x_key_store_open_key`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.348.1 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

If `keyObject == NULL`, then subsequent key injection does not use any KEK.

Parameters

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

Returns

The sss status.

Returns

The sss status.

10.15.4.349 Function `sss_se05x_key_store_save`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.349.1 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.

10.15.4.350 Function `sss_se05x_key_store_set_key`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.350.1 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.

Parameters

- **keyStore** – Key store context
- **keyObject** – Reference to a key and its 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

Returns**10.15.4.351 Function sss_se05x_mac_context_free**

- Defined in [File fsl_sss_se05x/apis.h](#)

10.15.4.351.1 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.

10.15.4.352 Function sss_se05x_mac_context_init

- Defined in [File fsl_sss_se05x/apis.h](#)

10.15.4.352.1 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.

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 values

- **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.

Returns

Status of the operation

10.15.4.353 Function `sss_se05x_mac_finish`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.353.1 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.

Parameters

- `context` – Pointer to mac context.
- `mac` – Output message MAC
- `macLen` – Computed MAC byte length

Return values

- `kStatus_SSS_Success` – The operation has completed successfully.
- `kStatus_SSS_Fail` – The operation has failed.

Returns

Status of the operation

10.15.4.354 Function `sss_se05x_mac_init`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.354.1 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.

Parameters

- `context` – Pointer to mac context.

Return values

- `kStatus_SSS_Success` – The operation has completed successfully.
- `kStatus_SSS_Fail` – The operation has failed.

Returns

Status of the operation

10.15.4.355 Function `sss_se05x_mac_one_go`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.355.1 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.

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 values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

10.15.4.356 Function `sss_se05x_mac_update`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.356.1 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.

Parameters

- **context** – Pointer to mac context.
- **message** – Buffer with a message chunk.
- **messageLen** – Length of the input buffer in bytes.

Return values

- **kStatus_SSS_Success** – The operation has completed successfully.
- **kStatus_SSS_Fail** – The operation has failed.

Returns

Status of the operation

10.15.4.357 Function `sss_se05x_mac_validate_one_go`

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.4.357.1 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

10.15.4.358 Function `sss_se05x_refresh_session`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.358.1 Function Documentation

`sss_status_t sss_se05x_refresh_session(sss_se05x_session_t *session, void *connectionData)`

Refreshes the SE05x session

10.15.4.359 Function `sss_se05x_rng_context_free`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.359.1 Function Documentation

`sss_status_t sss_se05x_rng_context_free(sss_se05x_rng_context_t *context)`

free random generator context.

Parameters

`context` – generator context.

Returns

status

10.15.4.360 Function `sss_se05x_rng_context_init`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.360.1 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);%%CRLF  
SP%%Now: Parameters are swapped  
sss_status_t sss_rng_context_init(  
    sss_rng_context_t *context, sss_session_t *session);
```

Parameters

- `session` – Session context.
- `context` – random generator context.

Returns

status

10.15.4.361 Function `sss_se05x_rng_get_random`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.361.1 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.

Parameters

- **context** – random generator context.
- **random_data** – buffer to hold random data.
- **dataLen** – required random number length

Returns

status

10.15.4.362 Function sss_se05x_session_close

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.362.1 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.

10.15.4.363 Function sss_se05x_session_create

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.363.1 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 application has to call [sss_session_create](#) before calling [sss_session_open](#).

Parameters

- **session** – [inout] Pointer to session context
- **subsystem** – [in] See [sss_session_open](#)
- **application_id** – [in] See [sss_session_open](#)
- **connection_type** – [in] See [sss_session_open](#)
- **connectionData** – [in] See [sss_session_open](#)

10.15.4.364 Function sss_se05x_session_delete

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.364.1 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.

10.15.4.365 Function `sss_se05x_session_open`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.365.1 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.

Note: `sss_session_open()` must not be called concurrently from multiple threads. The application must ensure this.

Parameters

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

Returns

`status`

10.15.4.366 Function `sss_se05x_session_prop_get_au8`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.366.1 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.

Parameters

- `session` – **[in]** Session context
- `property` – **[in]** Value that is part of [sss_session_prop_aus_t](#)
- `pValue` – **[out]** Output buffer array
- `pValueLen` – **[inout]** Count of values there are/must be read

Returns

10.15.4.367 Function `sss_se05x_session_prop_get_u32`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.367.1 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](#)

Parameters

- `session` – [in] Session context
- `property` – [in] Value that is part of [sss_session_prop_u32_t](#)
- `pValue` – [out]

Returns

10.15.4.368 Function `sss_se05x_set_feature`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.368.1 Function Documentation

`sss_status_t sss_se05x_set_feature(sss_se05x_session_t *session, SE05x_Applet_Feature_t feature, SE05x_Applet_Feature_Disable_t disable_features)`

Set features of the Applet.

See [Se05x_API_SetAppletFeatures](#)

10.15.4.369 Function `sss_se05x_symmetric_context_free`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.369.1 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.

10.15.4.370 Function `sss_se05x_symmetric_context_init`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.4.370.1 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.

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 values

- **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.

Returns

Status of the operation

10.15.4.371 Function `sss_se05x_tunnel`

- Defined in [File fsl_sss_se05x/apis.h](#)

10.15.4.371.1 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.

10.15.4.372 Function `sss_se05x_tunnel_context_free`

- Defined in [File fsl_sss_se05x/apis.h](#)

10.15.4.372.1 Function Documentation

```
void sss_se05x_tunnel_context_free(sss_se05x_tunnel_context_t *context)
```

Destructor for the tunnelling service context.

Parameters

context – **[out]** Pointer to tunnel context.

10.15.4.373 Function `sss_se05x_tunnel_context_init`

- Defined in [File fsl_sss_se05x/apis.h](#)

10.15.4.373.1 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

- **context – [out]** Pointer to tunnel context. Tunnel context is updated on function return.
- **session –** Pointer to session this tunnelling service belongs to.

10.15.4.374 Function `sss_session_close`

- Defined in [File fsl_sss_api.h](#)

10.15.4.374.1 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.

10.15.4.375 Function `sss_session_create`

- Defined in [File fsl_sss_api.h](#)

10.15.4.375.1 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 application has to call [sss_session_create](#) before calling [sss_session_open](#).

Parameters

- **session – [inout]** Pointer to session context
- **subsystem – [in]** See [sss_session_open](#)
- **application_id – [in]** See [sss_session_open](#)
- **connection_type – [in]** See [sss_session_open](#)
- **connectionData – [in]** See [sss_session_open](#)

10.15.4.376 Function `sss_session_delete`

- Defined in [File fsl_sss_api.h](#)

10.15.4.376.1 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.

10.15.4.377 Function `sss_session_open`

- Defined in [File fsl_sss_api.h](#)

10.15.4.377.1 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.

Note: `sss_session_open()` must not be called concurrently from multiple threads. The application must ensure this.

Parameters

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

Returns

`status`

10.15.4.378 Function `sss_session_prop_get_u8`

- Defined in [File fsl_sss_api.h](#)

10.15.4.378.1 Function Documentation

`sss_status_t sss_session_prop_get_u8(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.

Parameters

- `session` – **[in]** Session context
- `property` – **[in]** Value that is part of [sss_session_prop_u8_t](#)
- `pValue` – **[out]** Output buffer array
- `pValueLen` – **[inout]** Count of values there are/must be read

Returns

10.15.4.379 Function `sss_session_prop_get_u32`

- Defined in [File fsl_sss_api.h](#)

10.15.4.379.1 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](#)

Parameters

- `session` – **[in]** Session context
- `property` – **[in]** Value that is part of [sss_session_prop_u32_t](#)
- `pValue` – **[out]**

Returns

10.15.4.380 Function `sss_status_sz`

- Defined in [File fsl_sss_api.h](#)

10.15.4.380.1 Function Documentation

`const char *sss_status_sz(sss_status_t status)`

Returns string error code for [sss_status_t](#).

Parameters

- `status` – **[in]** See [sss_status_t](#)

Returns

String conversion of `status` to String.

10.15.4.381 Function `sss_symmetric_context_free`

- Defined in [File fsl_sss_api.h](#)

10.15.4.381.1 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.

10.15.4.382 Function `sss_symmetric_context_init`

- Defined in [File fsl_sss_api.h](#)

10.15.4.382.1 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.

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 values

- `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.

Returns

Status of the operation

10.15.4.383 Function sss_tunnel

- Defined in [File fsl_sss_api.h](#)

10.15.4.383.1 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

- `context` – [inout] 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.

10.15.4.384 Function sss_tunnel_context_free

- Defined in [File fsl_sss_api.h](#)

10.15.4.384.1 Function Documentation

`void sss_tunnel_context_free(sss_tunnel_t *context)`

Destructor for the tunnelling service context.

Parameters

- `context` – [out] Pointer to tunnel context.

10.15.4.385 Function sss_tunnel_context_init

- Defined in [File fsl_sss_api.h](#)

10.15.4.385.1 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

- context – [out]** Pointer to tunnel context. Tunnel context is updated on function return.
- session –** Pointer to session this tunnelling service belongs to.

10.15.4.386 Function tlvGet_Result

- Defined in [File se05x_tlv.h](#)

10.15.4.386.1 Function Documentation

`int tlvGet_Result(uint8_t *buf, size_t *pBufIndex, size_t bufLen, SE05x_TAG_t tag, SE05x_Result_t *presult)`

10.15.4.387 Function tlvGet_Se05xSession

- Defined in [File se05x_tlv.h](#)

10.15.4.387.1 Function Documentation

`int tlvGet_Se05xSession(uint8_t *buf, size_t *pBufIndex, const size_t bufLen, SE05x_TAG_t tag, pSe05xSession_t *pSessionId)`

10.15.4.388 Function tlvGet_SecureObjectType

- Defined in [File se05x_tlv.h](#)

10.15.4.388.1 Function Documentation

`int tlvGet_SecureObjectType(uint8_t *buf, size_t *pBufIndex, size_t bufLen, SE05x_TAG_t tag, SE05x_SecObjTyp_t *pType)`

10.15.4.389 Function tlvGet_TimeStamp

- Defined in [File se05x_tlv.h](#)

10.15.4.389.1 Function Documentation

```
int tlvGet_TimeStamp(uint8_t *buf, size_t *pBufIndex, const size_t bufLen, SE05x\_TAG\_t tag, SE05x\_TimeStamp\_t *pTs)
```

10.15.4.390 Function tlvGet_U16

- Defined in [File se05x_tlv.h](#)

10.15.4.390.1 Function Documentation

```
int tlvGet_U16(uint8_t *buf, size_t *pBufIndex, const size_t bufLen, SE05x\_TAG\_t tag, uint16_t *pRsp)
```

10.15.4.391 Function tlvGet_U32

- Defined in [File se05x_tlv.h](#)

10.15.4.391.1 Function Documentation

```
int tlvGet_U32(uint8_t *buf, size_t *pBufIndex, const size_t bufLen, SE05x\_TAG\_t tag, uint32_t *pRsp)
```

10.15.4.392 Function tlvGet_U8

- Defined in [File se05x_tlv.h](#)

10.15.4.392.1 Function Documentation

```
int tlvGet_U8(uint8_t *buf, size_t *pBufIndex, const size_t bufLen, SE05x\_TAG\_t tag, uint8_t *pRsp)
```

10.15.4.393 Function tlvGet_u8buf

- Defined in [File se05x_tlv.h](#)

10.15.4.393.1 Function Documentation

```
int tlvGet_u8buf(uint8_t *buf, size_t *pBufIndex, const size_t bufLen, SE05x\_TAG\_t tag, uint8_t *rsp, size_t *pRspLen)
```

10.15.4.394 Function tlvGet_ValueIndex

- Defined in [File se05x_tlv.h](#)

10.15.4.394.1 Function Documentation

```
int tlvGet_ValueIndex(uint8_t *buf, size_t *pBufIndex, const size_t bufLen, SE05x\_TAG\_t tag)
```

10.15.4.395 Function tlvSet_ECCurve

- Defined in [File se05x_tlv.h](#)

10.15.4.395.1 Function Documentation

```
int tlvSet_ECCurve(uint8_t **buf, size_t *bufLen, SE05x_TAG_t tag, SE05x_ECCurve_t value)
```

10.15.4.396 Function tlvSet_KeyID

- Defined in [File se05x_tlv.h](#)

10.15.4.396.1 Function Documentation

```
int tlvSet_KeyID(uint8_t **buf, size_t *bufLen, SE05x_TAG_t tag, uint32_t keyID)
```

10.15.4.397 Function tlvSet_MaxAttempts

- Defined in [File se05x_tlv.h](#)

10.15.4.397.1 Function Documentation

```
int tlvSet_MaxAttempts(uint8_t **buf, size_t *bufLen, SE05x_TAG_t tag, uint16_t maxAttempts)
```

10.15.4.398 Function tlvSet_Se05xPolicy

- Defined in [File se05x_tlv.h](#)

10.15.4.398.1 Function Documentation

```
int tlvSet_Se05xPolicy(const char *description, uint8_t **buf, size_t *bufLen, SE05x_TAG_t tag, Se05xPolicy_t *policy)
```

10.15.4.399 Function tlvSet_U16

- Defined in [File se05x_tlv.h](#)

10.15.4.399.1 Function Documentation

```
int tlvSet_U16(uint8_t **buf, size_t *bufLen, SE05x_TAG_t tag, uint16_t value)
```

10.15.4.400 Function tlvSet_U16Optional

- Defined in [File se05x_tlv.h](#)

10.15.4.400.1 Function Documentation

```
int tlvSet_U16Optional(uint8_t **buf, size_t *bufLen, SE05x_TAG_t tag, uint16_t value)
```

10.15.4.401 Function tlvSet_U32

- Defined in [File se05x_tlv.h](#)

10.15.4.401.1 Function Documentation

```
int tlvSet_U32(uint8_t **buf, size_t *bufLen, SE05x\_TAG\_t tag, uint32_t value)
```

10.15.4.402 Function tlvSet_U64_size

- Defined in [File se05x_tlv.h](#)

10.15.4.402.1 Function Documentation

```
int tlvSet_U64_size(uint8_t **buf, size_t *bufLen, SE05x\_TAG\_t tag, uint64_t value, uint16_t size)
```

10.15.4.403 Function tlvSet_U8

- Defined in [File se05x_tlv.h](#)

10.15.4.403.1 Function Documentation

```
int tlvSet_U8(uint8_t **buf, size_t *bufLen, SE05x\_TAG\_t tag, uint8_t value)
```

10.15.4.404 Function tlvSet_u8buf

- Defined in [File se05x_tlv.h](#)

10.15.4.404.1 Function Documentation

```
int tlvSet_u8buf(uint8_t **buf, size_t *bufLen, SE05x\_TAG\_t tag, const uint8_t *cmd, size_t cmdLen)
```

10.15.4.405 Function tlvSet_u8buf_features

- Defined in [File se05x_tlv.h](#)

10.15.4.405.1 Function Documentation

```
int tlvSet_u8buf_features(uint8_t **buf, size_t *bufLen, SE05x\_TAG\_t tag, pSe05xAppletFeatures\_t appletVariant)
```

10.15.4.406 Function tlvSet_u8buf_I2CM

- Defined in [File se05x_tlv.h](#)

10.15.4.406.1 Function Documentation

```
int tlvSet_u8buf_I2CM(uint8_t **buf, size_t *bufLen, SE05x\_I2CM\_TAG\_t tag, const uint8_t *cmd, size_t cmdLen)
```

10.15.4.407 Function tlvSet_u8bufOptional

- Defined in [File se05x_tlv.h](#)

10.15.4.407.1 Function Documentation

```
int tlvSet_u8bufOptional(uint8_t **buf, size_t *bufLen, SE05x\_TAG\_t tag, const uint8_t *cmd, size_t cmdLen)
```

10.15.4.408 Function tlvSet_u8bufOptional_ByteShift

- Defined in [File se05x_tlv.h](#)

10.15.4.408.1 Function Documentation

```
int tlvSet_u8bufOptional_ByteShift(uint8_t **buf, size_t *bufLen, SE05x\_TAG\_t tag, const uint8_t *cmd, size_t cmdLen)
```

10.15.5 Variables

10.15.5.1 Variable length

- Defined in [File nxp_iot_agent_service.h](#)

10.15.5.1.1 Variable Documentation

uint32_t **length**

Total length of the stored data (incl. all fields of configuration_data_header_t).

10.15.5.2 Variable number_of_services

- Defined in [File nxp_iot_agent_service.h](#)

10.15.5.2.1 Variable Documentation

uint32_t **number_of_services**

Number of stored service configurations.

10.15.5.3 Variable version

- Defined in [File nxp_iot_agent_service.h](#)

10.15.5.3.1 Variable Documentation

uint8_t [_sub_component_metaData_t::version](#)[2]

Version information of this subcomponent. Version information major.minor, according to java card virtual machine package versions, both values in range 0 to 127. byte 0 is Major value, byte 1 is Minor value.

10.15.6 Defines

10.15.6.1 Define CLAIMCODE_OBJ_ID

- Defined in [File nxp_iot_agent.h](#)

10.15.6.1.1 Define Documentation

CLAIMCODE_OBJ_ID

The keyid on the PSA API to use for fetching a claimcode.

10.15.6.2 Define DO_LOG_A

- Defined in [File se05x_tlv.h](#)

10.15.6.2.1 Define Documentation

DO_LOG_A(*TAG, DESCRIPTION, ARRAY, ARRAY_LEN*)

10.15.6.3 Define DO_LOG_V

- Defined in [File se05x_tlv.h](#)

10.15.6.3.1 Define Documentation

DO_LOG_V(*TAG, DESCRIPTION, VALUE*)

10.15.6.4 Define EDGELOCK2GO_KEYSTORE_ID

- Defined in [File nxp_iot_agent.h](#)

10.15.6.4.1 Define Documentation

EDGELOCK2GO_KEYSTORE_ID

The keystore that stores the credentials for the EdgeLock 2GO cloud service.

When connecting to the EdgeLock 2GO cloud service, for the client authentication a private key is required. This key is expected to be in a keystore which is registered to the EdgeLock 2GO agent.

In case there is a datastore which holds the information how to connect to the EdgeLock 2GO cloud service this datastore also holds the information where to get the private key.

In case the EdgeLock 2GO agent needs to fall back to compile-time constant connection information, it does assume that the private key for the client authentication is stored in a keystore that is registered with this ID.

10.15.6.5 Define ENABLE_DEPRECATED_API_WritePCR

- Defined in [File se05x_APDU_apis.h](#)

10.15.6.5.1 Define Documentation

ENABLE_DEPRECATED_API_WritePCR

Enable compilation of deprecated API Se05x_API_WritePCR Deprecated from Q1 2021. Support will be removed by Q1 2022

10.15.6.6 Define ENSURE_OR_BREAK

- Defined in [File nxEnsure.h](#)

10.15.6.6.1 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;
}
```

10.15.6.7 Define ENSURE_OR_EXIT_WITH_STATUS_ON_ERROR

- Defined in [File nxEnsure.h](#)

10.15.6.7.1 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;
    ...%%CRLF
    SP% value = Operation1();
    ENSURE_OR_QUIT_WITH_STATUS_ON_ERROR(0 == value, status, ERR_FAIL);%%CRLF
    SP% value = Operation2();
    ENSURE_OR_QUIT_WITH_STATUS_ON_ERROR(0 == value, status, ERR_NOT_ENOUGH_SPACE);%%CRLF
    SP% ...
    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.

10.15.6.8 Define ENSURE_OR_GO_CLEANUP

- Defined in [File nxEnsure.h](#)

10.15.6.8.1 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;
}
```

10.15.6.9 Define ENSURE_OR_GO_EXIT

- Defined in [File nxEnsure.h](#)

10.15.6.9.1 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);

...
exit:
return status;
}
```

10.15.6.10 Define ENSURE_OR_RETURN

- Defined in [File nxEnsure.h](#)

10.15.6.10.1 Define Documentation

ENSURE_OR_RETURN(CONDITION)

```
If condition fails, return
void SomeAPI()
{
...%%CRLF
SP%% status = Operation1();
ENSURE\_OR\_RETURN(0 == status);%%CRLF
SP%% status = Operation2();
ENSURE\_OR\_RETURN(0 == status);%%CRLF
```

```
SP%% ...%%CRLF  
SP%% return;  
}
```

Warning:

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

10.15.6.11 Define ENSURE_OR_RETURN_ON_ERROR

- Defined in [File nxEnsure.h](#)

10.15.6.11.1 Define Documentation

ENSURE_OR_RETURN_ON_ERROR(CONDITION, RETURN_VALUE)

If condition fails, return

```
int SomeAPI()  
{  
    ...%%CRLF  
    SP%% status = Operation1();  
ENSURE\_OR\_RETURN\_ON\_ERROR(0 == status, ERR_FAIL);%%CRLF  
    SP%% status = Operation2();  
ENSURE\_OR\_RETURN\_ON\_ERROR(0 == status, ERR_NOT_ENOUGH_SPACE);%%CRLF  
    SP%% ...%%CRLF  
    SP%% 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.

10.15.6.12 Define EX_SSS_BOOT_OPEN_HOST_SESSION

- Defined in [File ex_sss_main_inc.h](#)

10.15.6.12.1 Define Documentation

EX_SSS_BOOT_OPEN_HOST_SESSION

10.15.6.13 Define EX_SSS_BOOT_RTOS_STACK_SIZE

- Defined in [File ex_sss_main_inc.h](#)

10.15.6.13.1 Define Documentation

EX_SSS_BOOT_RTOS_STACK_SIZE

10.15.6.14 Define IOT_AGENT_CONFIGURATION_DATA_VERSION

- Defined in [File nxp_iot_agent_service.h](#)

10.15.6.14.1 Define Documentation

`IOT_AGENT_CONFIGURATION_DATA_VERSION`

10.15.6.15 Define `kSE05x_AuthType_AESKey`

- Defined in [File nxScp03_Types.h](#)

10.15.6.15.1 Define Documentation

`kSE05x_AuthType_AESKey`

10.15.6.16 Define `kSE05x_AuthType_ECKey`

- Defined in [File nxScp03_Types.h](#)

10.15.6.16.1 Define Documentation

`kSE05x_AuthType_ECKey`

10.15.6.17 Define `kSE05x_AuthType_None`

- Defined in [File nxScp03_Types.h](#)

10.15.6.17.1 Define Documentation

`kSE05x_AuthType_None`

10.15.6.18 Define `kSE05x_AuthType_SC03`

- Defined in [File nxScp03_Types.h](#)

10.15.6.18.1 Define Documentation

`kSE05x_AuthType_SC03`

10.15.6.19 Define `kSE05x_AuthType_UserID`

- Defined in [File nxScp03_Types.h](#)

10.15.6.19.1 Define Documentation

`kSE05x_AuthType_UserID`

10.15.6.20 Define `kSE05x_CLA`

- Defined in [File se05x_tlv.h](#)

10.15.6.20.1 Define Documentation

kSE05x_CLA

10.15.6.21 Define kSE05x_INS_I2CM_Attestation

- Defined in [File_se05x_enums.h](#)

10.15.6.21.1 Define Documentation

kSE05x_INS_I2CM_Attestation

When we want to read I2CM Data with attestation

10.15.6.22 Define kSE05x_INS_READ_With_Attestation

- Defined in [File_se05x_enums.h](#)

10.15.6.22.1 Define Documentation

kSE05x_INS_READ_With_Attestation

When we want to read with attestation

10.15.6.23 Define kSSS_AuthType_AppletSCP03

- Defined in [File_nxScp03_Types.h](#)

10.15.6.23.1 Define Documentation

kSSS_AuthType_AppletSCP03

10.15.6.24 Define kSSS_AuthType_FastSCP

- Defined in [File_nxScp03_Types.h](#)

10.15.6.24.1 Define Documentation

kSSS_AuthType_FastSCP

10.15.6.25 Define kSSS_AuthType_FastSCP_Counter

- Defined in [File_nxScp03_Types.h](#)

10.15.6.25.1 Define Documentation

kSSS_AuthType_FastSCP_Counter

10.15.6.26 Define kSSS_AuthType_INT_FastSCP_Counter

- Defined in [File_nxScp03_Types.h](#)

10.15.6.26.1 Define Documentation

kSSS_AuthType_INT_FastSCP_Counter

10.15.6.27 Define NX_ENSURE_DO_LOG_MESSAGE

- Defined in [File nxEnsure.h](#)

10.15.6.27.1 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.

10.15.6.28 Define NX_ENSURE_MESSAGE

- Defined in [File nxEnsure.h](#)

10.15.6.28.1 Define Documentation

NX_ENSURE_MESSAGE(strCONDITION)

Waring print of the parameter strCONDITION

Warning:

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

10.15.6.29 Define NXP_IOT_AGENT_EDGELOCK2GO_CLIENT_CERTIFICATE_BUFFER_SIZE

- Defined in [File nxp_iot_agent_utils.h](#)

10.15.6.29.1 Define Documentation

NXP_IOT_AGENT_EDGELOCK2GO_CLIENT_CERTIFICATE_BUFFER_SIZE

The size of the buffer to reserve for the EdgeLock 2GO cloud service client certificate.

10.15.6.30 Define PCONTEXT

- Defined in [File ex_sss_main_inc.h](#)

10.15.6.30.1 Define Documentation

PCONTEXT

10.15.6.31 Define SE050_INS_MASK_INS_CHAR

- Defined in [File se05x_enums.h](#)

10.15.6.31.1 Define Documentation

SE050_INS_MASK_INS_CHAR

3 MSBit for instruction characteristics.

10.15.6.32 Define SE050_INS_MASK_INSTRUCTION

- Defined in [File se05xEnums.h](#)

10.15.6.32.1 Define Documentation

SE050_INS_MASK_INSTRUCTION

5 LSBit for instruction

10.15.6.33 Define SE050_MAX_APDU_PAYLOAD_LENGTH

- Defined in [File se05xEnums.h](#)

10.15.6.33.1 Define Documentation

SE050_MAX_APDU_PAYLOAD_LENGTH

the maximum APDU payload length will be smaller, depending on which protocol applies, etc.

10.15.6.34 Define SE050_MAX_I2CM_COMMAND_LENGTH

- Defined in [File se05xEnums.h](#)

10.15.6.34.1 Define Documentation

SE050_MAX_I2CM_COMMAND_LENGTH

How many bytes can be used for buffer for I2C Master interface

10.15.6.35 Define SE050_MAX_NUMBER_OF_SESSIONS

- Defined in [File se05xEnums.h](#)

10.15.6.35.1 Define Documentation

SE050_MAX_NUMBER_OF_SESSIONS

Maximum number of session supported by SE050

10.15.6.36 Define SE050_OBJECT_IDENTIFIER_SIZE

- Defined in [File se05xEnums.h](#)

10.15.6.36.1 Define Documentation

SE050_OBJECT_IDENTIFIER_SIZE

Maximum number of session supported by SE050

10.15.6.37 Define Se05x_API_ECGenSharedSecret

- Defined in [File se05x_APDU.h](#)

10.15.6.37.1 Define Documentation

Se05x_API_ECGenSharedSecret

Wrapper for [Se05x_API_ECDHGenerateSharedSecret](#)

10.15.6.38 Define Se05x_API_SHAOneShot

- Defined in [File se05x_APDU.h](#)

10.15.6.38.1 Define Documentation

Se05x_API_SHAOneShot

Wrapper for [Se05x_API_DigestOneShot](#)

10.15.6.39 Define Se05x_API_WriteECKey_with_version

- Defined in [File se05x_04_xx_APDU_apis.h](#)

10.15.6.39.1 Define Documentation

Se05x_API_WriteECKey_with_version

Se05x_API_WriteECKey_with_version See [Se05x_API_WriteECKey_Ver](#)

10.15.6.40 Define se05x_auth_context_t

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.6.40.1 Define Documentation

se05x_auth_context_t

deprecated : Used only for backwards compatibility

10.15.6.41 Define SE05x_AuthCtx_t

- Defined in [File nxScp03_Types.h](#)

10.15.6.41.1 Define Documentation

SE05x_AuthCtx_t

10.15.6.42 Define SE05x_AuthType_t

- Defined in [File nxScp03_Types.h](#)

10.15.6.42.1 Define Documentation

SE05x_AuthType_t

10.15.6.43 Define SE05x_Connect_Ctx_t

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.6.43.1 Define Documentation

SE05x_Connect_Ctx_t

deprecated : Used only for backwards compatibility

10.15.6.44 Define SE05x_CryptoObjectID_t

- Defined in [File se05xEnums.h](#)

10.15.6.44.1 Define Documentation

SE05x_CryptoObjectID_t

Crypto object identifiers

10.15.6.45 Define SE05x_KeyID_KEK_NONE

- Defined in [File se05xEnums.h](#)

10.15.6.45.1 Define Documentation

SE05x_KeyID_KEK_NONE

Case when there is no KEK

10.15.6.46 Define SE05x_KeyID_MFDF_NONE

- Defined in [File se05xEnums.h](#)

10.15.6.46.1 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].

10.15.6.47 Define SE05x_MaxAttempts_NA

- Defined in [File se05xEnums.h](#)

10.15.6.47.1 Define Documentation

SE05x_MaxAttempts_NA

Identify in code that this is not an AUTH object and hence not applicable

10.15.6.48 Define SE05x_MaxAttempts_UNLIMITED

- Defined in [File se05x_enums.h](#)

10.15.6.48.1 Define Documentation

SE05x_MaxAttempts_UNLIMITED

Fall back to applet default

10.15.6.49 Define SEMS_LITE_AGENT_PROD_NAME

- Defined in [File sems_lite_api_ver.h](#)

10.15.6.49.1 Define Documentation

SEMS_LITE_AGENT_PROD_NAME

10.15.6.50 Define SEMS_LITE_AGENT_PROD_NAME_VER_FULL

- Defined in [File sems_lite_api_ver.h](#)

10.15.6.50.1 Define Documentation

SEMS_LITE_AGENT_PROD_NAME_VER_FULL

10.15.6.51 Define SEMS_LITE_AGENT_VER_DEV

- Defined in [File sems_lite_api_ver.h](#)

10.15.6.51.1 Define Documentation

SEMS_LITE_AGENT_VER_DEV

10.15.6.52 Define SEMS_LITE_AGENT_VER_MAJOR

- Defined in [File sems_lite_api_ver.h](#)

10.15.6.52.1 Define Documentation

SEMS_LITE_AGENT_VER_MAJOR

10.15.6.53 Define SEMS_LITE_AGENT_VER_MAJOR_MINOR

- Defined in [File sems_lite_api_ver.h](#)

10.15.6.53.1 Define Documentation

SEMS_LITE_AGENT_VER_MAJOR_MINOR

10.15.6.54 Define SEMS_LITE_AGENT_VER_MAJOR_MINOR_DEV

- Defined in [File_sems_lite_api_ver.h](#)

10.15.6.54.1 Define Documentation**SEMS_LITE_AGENT_VER_MAJOR_MINOR_DEV****10.15.6.55 Define SEMS_LITE_AGENT_VER_MINOR**

- Defined in [File_sems_lite_api_ver.h](#)

10.15.6.55.1 Define Documentation**SEMS_LITE_AGENT_VER_MINOR****10.15.6.56 Define SEMS_LITE_AGENT_VER_STRING_NUM**

- Defined in [File_sems_lite_api_ver.h](#)

10.15.6.56.1 Define Documentation**SEMS_LITE_AGENT_VER_STRING_NUM****10.15.6.57 Define SEMS_LITE_API_VERSION_MAJOR**

- Defined in [File_sems_lite_api.h](#)

10.15.6.57.1 Define Documentation**SEMS_LITE_API_VERSION_MAJOR****10.15.6.58 Define SEMS_LITE_API_VERSION_MINOR**

- Defined in [File_sems_lite_api.h](#)

10.15.6.58.1 Define Documentation**SEMS_LITE_API_VERSION_MINOR****10.15.6.59 Define SEMS_LITE_API_VERSION_PATCH**

- Defined in [File_sems_lite_api.h](#)

10.15.6.59.1 Define Documentation**SEMS_LITE_API_VERSION_PATCH**

10.15.6.60 Define SEMS_LITE_GET_DATA_CMD_BUF_LEN

- Defined in [File sems_lite_api.h](#)

10.15.6.60.1 Define Documentation

SEMS_LITE_GET_DATA_CMD_BUF_LEN

10.15.6.61 Define SEMS_LITE_PKG_VERSION_LEN

- Defined in [File sems_lite_api.h](#)

10.15.6.61.1 Define Documentation

SEMS_LITE_PKG_VERSION_LEN

10.15.6.62 Define sss_aead_context_free

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.62.1 Define Documentation

sss_aead_context_free(*context*)

10.15.6.63 Define sss_aead_context_init

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.63.1 Define Documentation

sss_aead_context_init(*context*, *session*, *keyObject*, *algorithm*, *mode*)

10.15.6.64 Define sss_aead_finish

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.64.1 Define Documentation

sss_aead_finish(*context*, *srcData*, *srcLen*, *destData*, *destLen*, *tag*, *tagLen*)

10.15.6.65 Define sss_aead_init

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.65.1 Define Documentation

sss_aead_init(*context*, *nonce*, *nonceLen*, *tagLen*, *aadLen*, *payloadLen*)

10.15.6.66 Define `sss_aead_one_go`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.66.1 Define Documentation

`sss_aead_one_go(context, srcData, destData, size, nonce, nonceLen, aad, aadLen, tag, tagLen)`

10.15.6.67 Define `SSS_AEAD_TYPE_IS_SE05X`

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.6.67.1 Define Documentation

`SSS_AEAD_TYPE_IS_SE05X(context)`

Are we using SE05X as crypto subsystem?

10.15.6.68 Define `sss_aead_update`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.68.1 Define Documentation

`sss_aead_update(context, srcData, srcLen, destData, destLen)`

10.15.6.69 Define `sss_aead_update_aad`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.69.1 Define Documentation

`sss_aead_update_aad(context, aadData, aadDataLen)`

10.15.6.70 Define `SSS_AES_BLOCK_SIZE`

- Defined in [File fsl_sss_api.h](#)

10.15.6.70.1 Define Documentation

`SSS_AES_BLOCK_SIZE`

Size of an AES Block, in bytes

10.15.6.71 Define `SSS_API_VERSION`

- Defined in [File fsl_sss_api.h](#)

10.15.6.71.1 Define Documentation

`SSS_API_VERSION`

Version of the SSS API

10.15.6.72 Define `sss_asymmetric_context_free`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.72.1 Define Documentation

`sss_asymmetric_context_free(context)`

10.15.6.73 Define `sss_asymmetric_context_init`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.73.1 Define Documentation

`sss_asymmetric_context_init(context, session, keyObject, algorithm, mode)`

10.15.6.74 Define `sss_asymmetric_decrypt`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.74.1 Define Documentation

`sss_asymmetric_decrypt(context, srcData, srcLen, destData, destLen)`

10.15.6.75 Define `sss_asymmetric_encrypt`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.75.1 Define Documentation

`sss_asymmetric_encrypt(context, srcData, srcLen, destData, destLen)`

10.15.6.76 Define `sss_asymmetric_sign_digest`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.76.1 Define Documentation

`sss_asymmetric_sign_digest(context, digest, digestLen, signature, signatureLen)`

10.15.6.77 Define `SSS_ASYMMETRIC_TYPE_IS_SE05X`

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.6.77.1 Define Documentation

`SSS_ASYMMETRIC_TYPE_IS_SE05X(context)`

Are we using SE05X as crypto subsystem?

10.15.6.78 Define sss_asymmetric_verify_digest

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.78.1 Define Documentation

`sss_asymmetric_verify_digest(context, digest, digestLen, signature, signatureLen)`

10.15.6.79 Define sss_cipher_crypt_ctr

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.79.1 Define Documentation

`sss_cipher_crypt_ctr(context, srcData, destData, size, initialCounter, lastEncryptedCounter, szLeft)`

10.15.6.80 Define sss_cipher_finish

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.80.1 Define Documentation

`sss_cipher_finish(context, srcData, srcLen, destData, destLen)`

10.15.6.81 Define sss_cipher_init

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.81.1 Define Documentation

`sss_cipher_init(context, iv, ivLen)`

10.15.6.82 Define sss_cipher_one_go

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.82.1 Define Documentation

`sss_cipher_one_go(context, iv, ivLen, srcData, destData, dataLen)`

10.15.6.83 Define sss_cipher_one_go_v2

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.83.1 Define Documentation

`sss_cipher_one_go_v2(context, iv, ivLen, srcData, srcLen, destData, pDataLen)`

10.15.6.84 Define sss_cipher_update

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.84.1 Define Documentation

`sss_cipher_update(context, srcData, srcLen, destData, destLen)`

10.15.6.85 Define sss_derive_key_context_free

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.85.1 Define Documentation

`sss_derive_key_context_free(context)`

10.15.6.86 Define sss_derive_key_context_init

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.86.1 Define Documentation

`sss_derive_key_context_init(context, session, keyObject, algorithm, mode)`

10.15.6.87 Define sss_derive_key_dh

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.87.1 Define Documentation

`sss_derive_key_dh(context, otherPartyKeyObject, derivedKeyObject)`

10.15.6.88 Define sss_derive_key_go

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.88.1 Define Documentation

`sss_derive_key_go(context, saltData, saltLen, info, infoLen, derivedKeyObject, deriveDataLen, hkdfOutput, hkdfOutputLen)`

10.15.6.89 Define SSS_DERIVE_KEY_TYPE_IS_SE05X

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.6.89.1 Define Documentation

`SSS_DERIVE_KEY_TYPE_IS_SE05X(context)`

Are we using SE05X as crypto subsystem?

10.15.6.90 Define SSS_DES_BLOCK_SIZE

- Defined in [File fsl_sss_api.h](#)

10.15.6.90.1 Define Documentation

SSS_DES_BLOCK_SIZE

Size of a DES Block, in bytes

10.15.6.91 Define SSS_DES_IV_SIZE

- Defined in [File fsl_sss_api.h](#)

10.15.6.91.1 Define Documentation

SSS_DES_IV_SIZE

Size of a DES IV, in bytes

10.15.6.92 Define SSS_DES_KEY_SIZE

- Defined in [File fsl_sss_api.h](#)

10.15.6.92.1 Define Documentation

SSS_DES_KEY_SIZE

Size of a DES Key, in bytes

10.15.6.93 Define sss_digest_context_free

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.93.1 Define Documentation

sss_digest_context_free(context)

10.15.6.94 Define sss_digest_context_init

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.94.1 Define Documentation

sss_digest_context_init(context, session, algorithm, mode)

10.15.6.95 Define sss_digest_finish

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.95.1 Define Documentation

`sss_digest_finish(context, digest, digestLen)`

10.15.6.96 Define `sss_digest_init`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.96.1 Define Documentation

`sss_digest_init(context)`

10.15.6.97 Define `sss_digest_one_go`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.97.1 Define Documentation

`sss_digest_one_go(context, message, messageLen, digest, digestLen)`

10.15.6.98 Define `SSS_DIGEST_TYPE_IS_SE05X`

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.6.98.1 Define Documentation

`SSS_DIGEST_TYPE_IS_SE05X(context)`

Are we using SE05X as crypto subsystem?

10.15.6.99 Define `sss_digest_update`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.99.1 Define Documentation

`sss_digest_update(context, message, messageLen)`

10.15.6.100 Define `SSS_ENUM`

- Defined in [File fsl_sss_api.h](#)

10.15.6.100.1 Define Documentation

`SSS_ENUM(GROUP, INDEX)`

Helper macro to set enum value

10.15.6.101 Define `sss_key_object_allocate_handle`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.101.1 Define Documentation

`sss_key_object_allocate_handle(keyObject, keyId, keyPart, cipherType, keyByteLenMax, options)`

10.15.6.102 Define `sss_key_object_free`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.102.1 Define Documentation

`sss_key_object_free(keyObject)`

10.15.6.103 Define `sss_key_object_get_access`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.103.1 Define Documentation

`sss_key_object_get_access(keyObject, access)`

10.15.6.104 Define `sss_key_object_get_handle`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.104.1 Define Documentation

`sss_key_object_get_handle(keyObject, keyId)`

10.15.6.105 Define `sss_key_object_get_purpose`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.105.1 Define Documentation

`sss_key_object_get_purpose(keyObject, purpose)`

10.15.6.106 Define `sss_key_object_get_user`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.106.1 Define Documentation

`sss_key_object_get_user(keyObject, user)`

10.15.6.107 Define `sss_key_object_init`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.107.1 Define Documentation

`sss_key_object_init(keyObject, keyStore)`

10.15.6.108 Define `sss_key_object_set_access`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.108.1 Define Documentation

`sss_key_object_set_access(keyObject, access, options)`

10.15.6.109 Define `sss_key_object_set_eccgfp_group`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.109.1 Define Documentation

`sss_key_object_set_eccgfp_group(keyObject, group)`

10.15.6.110 Define `sss_key_object_set_purpose`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.110.1 Define Documentation

`sss_key_object_set_purpose(keyObject, purpose, options)`

10.15.6.111 Define `sss_key_object_set_user`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.111.1 Define Documentation

`sss_key_object_set_user(keyObject, user, options)`

10.15.6.112 Define `sss_key_store_allocate`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.112.1 Define Documentation

`sss_key_store_allocate(keyStore, keyStoreId)`

10.15.6.113 Define `sss_key_store_context_free`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.113.1 Define Documentation

`sss_key_store_context_free(keyStore)`

10.15.6.114 Define `sss_key_store_context_init`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.114.1 Define Documentation

`sss_key_store_context_init(keyStore, session)`

10.15.6.115 Define `sss_key_store_erase_key`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.115.1 Define Documentation

`sss_key_store_erase_key(keyStore, keyObject)`

10.15.6.116 Define `sss_key_store_freeze_key`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.116.1 Define Documentation

`sss_key_store_freeze_key(keyStore, keyObject)`

10.15.6.117 Define `sss_key_store_generate_key`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.117.1 Define Documentation

`sss_key_store_generate_key(keyStore, keyObject, keyBitLen, options)`

10.15.6.118 Define `sss_key_store_get_key`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.118.1 Define Documentation

`sss_key_store_get_key(keyStore, keyObject, data, dataLen, pKeyBitLen)`

10.15.6.119 Define `sss_key_store_load`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.119.1 Define Documentation

`sss_key_store_load(keyStore)`

10.15.6.120 Define `sss_key_store_open_key`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.120.1 Define Documentation

`sss_key_store_open_key(keyStore, keyObject)`

10.15.6.121 Define `sss_key_store_save`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.121.1 Define Documentation

`sss_key_store_save(keyStore)`

10.15.6.122 Define `sss_key_store_set_key`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.122.1 Define Documentation

`sss_key_store_set_key(keyStore, keyObject, data, dataLen, keyBitLen, options, optionsLen)`

10.15.6.123 Define `SSS_KEY_STORE_TYPE_IS_SE05X`

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.6.123.1 Define Documentation

`SSS_KEY_STORE_TYPE_IS_SE05X(keyStore)`

Are we using SE05X as crypto subsystem?

10.15.6.124 Define `sss_mac_context_free`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.124.1 Define Documentation

`sss_mac_context_free(context)`

10.15.6.125 Define `sss_mac_context_init`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.125.1 Define Documentation

`sss_mac_context_init(context, session, keyObject, algorithm, mode)`

10.15.6.126 Define `sss_mac_finish`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.126.1 Define Documentation

`sss_mac_finish(context, mac, macLen)`

10.15.6.127 Define `sss_mac_init`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.127.1 Define Documentation

`sss_mac_init(context)`

10.15.6.128 Define `sss_mac_one_go`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.128.1 Define Documentation

`sss_mac_one_go(context, message, messageLen, mac, macLen)`

10.15.6.129 Define `SSS_MAC_TYPE_IS_SE05X`

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.6.129.1 Define Documentation

`SSS_MAC_TYPE_IS_SE05X(context)`

Are we using SE05X as crypto subsystem?

10.15.6.130 Define `sss_mac_update`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.130.1 Define Documentation

`sss_mac_update(context, message, messageLen)`

10.15.6.131 Define `SSS_OBJECT_TYPE_IS_SE05X`

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.6.131.1 Define Documentation

SSS_OBJECT_TYPE_IS_SE05X(*pObject*)

Are we using SE05X as crypto subsystem?

10.15.6.132 Define **sss_rng_context_free**

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.132.1 Define Documentation

sss_rng_context_free(*context*)

10.15.6.133 Define **sss_rng_context_init**

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.133.1 Define Documentation

sss_rng_context_init(*context, session*)

10.15.6.134 Define **SSS_RNG_CONTEXT_TYPE_IS_SE05X**

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.6.134.1 Define Documentation

SSS_RNG_CONTEXT_TYPE_IS_SE05X(*context*)

Are we using SE05X as crypto subsystem?

10.15.6.135 Define **sss_rng_get_random**

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.135.1 Define Documentation

sss_rng_get_random(*context, random_data, dataLen*)

10.15.6.136 Define **sss_session_close**

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.136.1 Define Documentation

sss_session_close(*session*)

10.15.6.137 Define **sss_session_create**

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.137.1 Define Documentation

`sss_session_create(session, subsystem, application_id, connection_type, connectionData)`

10.15.6.138 Define `sss_session_delete`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.138.1 Define Documentation

`sss_session_delete(session)`

10.15.6.139 Define `sss_session_open`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.139.1 Define Documentation

`sss_session_open(session, subsystem, application_id, connection_type, connectionData)`

10.15.6.140 Define `sss_session_prop_get_au8`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.140.1 Define Documentation

`sss_session_prop_get_au8(session, property, pValue, pValueLen)`

10.15.6.141 Define `sss_session_prop_get_u32`

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.141.1 Define Documentation

`sss_session_prop_get_u32(session, property, pValue)`

10.15.6.142 Define `SSS_SESSION_TYPE_IS_SE05X`

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.6.142.1 Define Documentation

`SSS_SESSION_TYPE_IS_SE05X(session)`

Are we using SE05X as crypto subsystem?

10.15.6.143 Define `SSS_SUBSYSTEM_TYPE_IS_SE05X`

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.6.143.1 Define Documentation

SSS_SUBSYSTEM_TYPE_IS_SE05X(*subsystem*)

Are we using SE05X as crypto subsystem?

10.15.6.144 Define **sss_symmetric_context_free**

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.144.1 Define Documentation

sss_symmetric_context_free(*context*)

10.15.6.145 Define **sss_symmetric_context_init**

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.145.1 Define Documentation

sss_symmetric_context_init(*context, session, keyObject, algorithm, mode*)

10.15.6.146 Define **SSS_SYMMETRIC_TYPE_IS_SE05X**

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.6.146.1 Define Documentation

SSS_SYMMETRIC_TYPE_IS_SE05X(*context*)

Are we using SE05X as crypto subsystem?

10.15.6.147 Define **sss_tunnel**

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.147.1 Define Documentation

sss_tunnel(*context, data, dataLen, keyObjects, keyObjectCount, tunnelType*)

10.15.6.148 Define **sss_tunnel_context_free**

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.148.1 Define Documentation

sss_tunnel_context_free(*context*)

10.15.6.149 Define **sss_tunnel_context_init**

- Defined in [File fsl_sss_se05x_apis.h](#)

10.15.6.149.1 Define Documentation

sss_tunnel_context_init(*context, session*)

10.15.6.150 Define SSS_TUNNEL_CONTEXT_TYPE_IS_SE05X

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.6.150.1 Define Documentation

SSS_TUNNEL_CONTEXT_TYPE_IS_SE05X(*context*)

Are we using SE05X as crypto subsystem?

10.15.6.151 Define SSS_TUNNEL_TYPE_IS_SE05X

- Defined in [File fsl_sss_se05x_types.h](#)

10.15.6.151.1 Define Documentation

SSS_TUNNEL_TYPE_IS_SE05X(*context*)

Are we using SE05X as crypto subsystem?

10.15.6.152 Define TLVSET_AttestationAlgo

- Defined in [File se05x_tlv.h](#)

10.15.6.152.1 Define Documentation

TLVSET_AttestationAlgo

See TLVSET_U8

10.15.6.153 Define TLVSET_CipherMode

- Defined in [File se05x_tlv.h](#)

10.15.6.153.1 Define Documentation

TLVSET_CipherMode

See TLVSET_U8

10.15.6.154 Define TLVSET_CryptoContext

- Defined in [File se05x_tlv.h](#)

10.15.6.154.1 Define Documentation

TLVSET_CryptoContext

See TLVSET_U8

10.15.6.155 Define TLVSET_CryptoModeSubType

- Defined in [File se05x_tlv.h](#)

10.15.6.155.1 Define Documentation

TLVSET_CryptoModeSubType(*DESCRIPTION*, *PBUF*, *PBUFLLEN*, *TAG*, *VALUE*)

See TLVSET_U8

10.15.6.156 Define TLVSET_CryptoObjectID

- Defined in [File se05x_tlv.h](#)

10.15.6.156.1 Define Documentation

TLVSET_CryptoObjectID

See TLVSET_U16

10.15.6.157 Define TLVSET_DigestMode

- Defined in [File se05x_tlv.h](#)

10.15.6.157.1 Define Documentation

TLVSET_DigestMode

See TLVSET_U8

10.15.6.158 Define TLVSET_ECCurve

- Defined in [File se05x_tlv.h](#)

10.15.6.158.1 Define Documentation

TLVSET_ECCurve(*DESCRIPTION*, *PBUF*, *PBUFLLEN*, *TAG*, *VALUE*)

Creates a TLV set with EC Curve value

10.15.6.159 Define TLVSET_ECCurveParam

- Defined in [File se05x_tlv.h](#)

10.15.6.159.1 Define Documentation

TLVSET_ECCurveParam

See TLVSET_U8

10.15.6.160 Define TLVSET_ECSignatureAlgo

- Defined in [File se05x_tlv.h](#)

10.15.6.160.1 Define Documentation

TLVSET_ECSignatureAlgo

See TLVSET_U8

10.15.6.161 Define TLVSET_EDSignatureAlgo

- Defined in [File se05x_tlv.h](#)

10.15.6.161.1 Define Documentation

TLVSET_EDSignatureAlgo

See TLVSET_U8

10.15.6.162 Define TLVSET_KeyID

- Defined in [File se05x_tlv.h](#)

10.15.6.162.1 Define Documentation

TLVSET_KeyID(*DESCRIPTION, PBUF, PBUFLEN, TAG, VALUE*)

Creates a TLV set with KeyID

10.15.6.163 Define TLVSET_MacOperation

- Defined in [File se05x_tlv.h](#)

10.15.6.163.1 Define Documentation

TLVSET_MacOperation

See TLVSET_U8

10.15.6.164 Define TLVSET_MaxAttempts

- Defined in [File se05x_tlv.h](#)

10.15.6.164.1 Define Documentation

TLVSET_MaxAttempts(*DESCRIPTION, PBUF, PBUFLEN, TAG, VALUE*)

Creates a TLV set with maximum attempts value

10.15.6.165 Define TLVSET_MemoryType

- Defined in [File se05x_tlv.h](#)

10.15.6.165.1 Define Documentation

TLVSET_MemoryType

See TLVSET_U8

10.15.6.166 Define TLVSET_PlatformSCPRequest

- Defined in [File se05x_tlv.h](#)

10.15.6.166.1 Define Documentation

TLVSET_PlatformSCPRequest

See TLVSET_U8

10.15.6.167 Define TLVSET_RSAEncryptionAlgo

- Defined in [File se05x_tlv.h](#)

10.15.6.167.1 Define Documentation

TLVSET_RSAEncryptionAlgo

See TLVSET_U8

10.15.6.168 Define TLVSET_RSAResKeyComponent

- Defined in [File se05x_tlv.h](#)

10.15.6.168.1 Define Documentation

TLVSET_RSAResKeyComponent

See TLVSET_U8

10.15.6.169 Define TLVSET_RSAPubKeyComp

- Defined in [File se05x_tlv.h](#)

10.15.6.169.1 Define Documentation

TLVSET_RSAPubKeyComp

See TLVSET_U8

10.15.6.170 Define TLVSET_RSASignatureAlgo

- Defined in [File se05x_tlv.h](#)

10.15.6.170.1 Define Documentation

TLVSET_RSASignatureAlgo

See TLVSET_U8

10.15.6.171 Define TLVSET_Se05xPolicy

- Defined in [File se05x_tlv.h](#)

10.15.6.171.1 Define Documentation

TLVSET_Se05xPolicy(*DESCRIPTION*, *PBUF*, *PBUFLLEN*, *TAG*, *POLICY*)

Creates a TLV set using parameters for SE05xPolicy information

10.15.6.172 Define TLVSET_Se05xSession

- Defined in [File se05x_tlv.h](#)

10.15.6.172.1 Define Documentation

TLVSET_Se05xSession(*DESCRIPTION*, *PBUF*, *PBUFLLEN*, *TAG*, *SESSIONID*)

Creates a TLV set for SE05x Session

10.15.6.173 Define TLVSET_U16

- Defined in [File se05x_tlv.h](#)

10.15.6.173.1 Define Documentation

TLVSET_U16(*DESCRIPTION*, *PBUF*, *PBUFLLEN*, *TAG*, *VALUE*)

Creates a TLV set with -bit unsigned integer value

10.15.6.174 Define TLVSET_U16Optional

- Defined in [File se05x_tlv.h](#)

10.15.6.174.1 Define Documentation

TLVSET_U16Optional(*DESCRIPTION*, *PBUF*, *PBUFLLEN*, *TAG*, *VALUE*)

Creates a TLV set with 16-bit unsigned integer value

10.15.6.175 Define TLVSET_U32

- Defined in [File se05x_tlv.h](#)

10.15.6.175.1 Define Documentation

TLVSET_U32(*DESCRIPTION*, *PBUF*, *PBUFLLEN*, *TAG*, *VALUE*)

Creates a TLV set with 32-bit unsigned integer value

10.15.6.176 Define TLVSET_U64_SIZE

- Defined in [File se05x_tlv.h](#)

10.15.6.176.1 Define Documentation

TLVSET_U64_SIZE(*DESCRIPTION, PBUF, PBUFLEN, TAG, VALUE, SIZE*)

Creates a TLV set with 64-bit unsigned integer value and a specified size

10.15.6.177 Define TLVSET_U8

- Defined in [File se05x_tlv.h](#)

10.15.6.177.1 Define Documentation

TLVSET_U8(*DESCRIPTION, PBUF, PBUFLEN, TAG, VALUE*)

Creates a TLV set with 8-bit unsigned integer value

10.15.6.178 Define TLVSET_u8buf

- Defined in [File se05x_tlv.h](#)

10.15.6.178.1 Define Documentation

TLVSET_u8buf(*DESCRIPTION, PBUF, PBUFLEN, TAG, CMD, CMDLEN*)

10.15.6.179 Define TLVSET_u8buf_I2CM

- Defined in [File se05x_tlv.h](#)

10.15.6.179.1 Define Documentation

TLVSET_u8buf_I2CM(*DESCRIPTION, PBUF, PBUFLEN, TAG, CMD, CMDLEN*)

10.15.6.180 Define TLVSET_u8bufOptional

- Defined in [File se05x_tlv.h](#)

10.15.6.180.1 Define Documentation

TLVSET_u8bufOptional(*DESCRIPTION, PBUF, PBUFLEN, TAG, CMD, CMDLEN*)

10.15.6.181 Define TLVSET_u8bufOptional_ByteShift

- Defined in [File se05x_tlv.h](#)

10.15.6.181.1 Define Documentation

TLVSET_u8bufOptional_ByteShift(*DESCRIPTION, PBUF, PBUFLEN, TAG, CMD, CMDLEN*)

10.15.6.182 Define TLVSET_Variant

- Defined in [File se05x_tlv.h](#)

10.15.6.182.1 Define Documentation

TLVSET_Variant

See TLVSET_U8

10.15.7 Typedefs

10.15.7.1 Typedef configuration_data_header_t

- Defined in [File nxp_iot_agent_service.h](#)

10.15.7.1.1 Typedef Documentation

`typedef __PACKED_STRUCT configuration_data_header_t`

The header of configuration data stored in a datastore.

10.15.7.2 Typedef get_endpoint_info_callback_t

- Defined in [File nxp_iot_agent_dispatcher.h](#)

10.15.7.2.1 Typedef Documentation

`typedef bool (*get_endpoint_info_callback_t)(void *context, void *endpoint_information)`

10.15.7.3 Typedef iot_agent_context_t

- Defined in [File nxp_iot_agent_dispatcher.h](#)

10.15.7.3.1 Typedef Documentation

`typedef struct iot_agent_context_t iot_agent_context_t`

10.15.7.4 Typedef iot_agent_datastore_allocator_t

- Defined in [File nxp_iot_agent_datastore.h](#)

10.15.7.4.1 Typedef Documentation

`typedef iot_agent_status_t (*iot_agent_datastore_allocator_t)(void *context, size_t len)`

10.15.7.5 Typedef iot_agent_datastore_committer_t

- Defined in [File nxp_iot_agent_datastore.h](#)

10.15.7.5.1 Typedef Documentation

`typedef iot_agent_status_t (*iot_agent_datastore_committer_t)(void *context)`

10.15.7.6 Typedef iot_agent_datastore_destroyer_t

- Defined in [File nxp_iot_agent_datastore.h](#)

10.15.7.6.1 Typedef Documentation

```
typedef iot_agent_status_t (*iot_agent_datastore_destroyer_t)(void *context)
```

10.15.7.7 Typedef iot_agent_datastore_reader_t

- Defined in [File nxp_iot_agent_datastore.h](#)

10.15.7.7.1 Typedef Documentation

```
typedef iot_agent_status_t (*iot_agent_datastore_reader_t)(void *context, void *dst, size_t offset, size_t *len)
```

10.15.7.8 Typedef iot_agent_datastore_validator_t

- Defined in [File nxp_iot_agent_datastore.h](#)

10.15.7.8.1 Typedef Documentation

```
typedef bool (*iot_agent_datastore_validator_t)(const iot_agent_datastore_t *context)
```

10.15.7.9 Typedef iot_agent_datastore_writer_t

- Defined in [File nxp_iot_agent_datastore.h](#)

10.15.7.9.1 Typedef Documentation

```
typedef iot_agent_status_t (*iot_agent_datastore_writer_t)(void *context, size_t offset, const void *src, size_t len)
```

10.15.7.10 Typedef iot_agent_keystore_destroyer_t

- Defined in [File nxp_iot_agent_keystore.h](#)

10.15.7.10.1 Typedef Documentation

```
typedef iot_agent_status_t (*iot_agent_keystore_destroyer_t)(void *context)
```

10.15.7.11 Typedef iot_agent_keystore_session_closer_t

- Defined in [File nxp_iot_agent_keystore.h](#)

10.15.7.11.1 Typedef Documentation

```
typedef void (*iot_agent_keystore_session_closer_t)(void *context)
```

10.15.7.12 Typedef `iot_agent_keystore_session_opener_t`

- Defined in [File nxp_iot_agent_keystore.h](#)

10.15.7.12.1 Typedef Documentation

```
typedef iot_agent_status_t (*iot_agent_keystore_session_opener_t)(void *context)
```

10.15.7.13 Typedef `nxp_iot_ResponsePayload`

- Defined in [File nxp_iot_datastore.h](#)

10.15.7.13.1 Typedef Documentation

```
typedef struct _nxp_iot_ResponsePayload nxp_iot_ResponsePayload
```

10.15.7.14 Typedef `nxp_iot_UpdateStatusReport`

- Defined in [File nxp_iot_agent.h](#)

10.15.7.14.1 Typedef Documentation

```
typedef struct _nxp_iot_UpdateStatusReport nxp_iot_UpdateStatusReport
```

10.15.7.15 Typedef `pb_field_t`

- Defined in [File nxp_iot_agent.h](#)

10.15.7.15.1 Typedef Documentation

```
typedef struct pb_field_s pb_field_t
```

10.15.7.16 Typedef `pb_istream_t`

- Defined in [File nxp_iot_agent.h](#)

10.15.7.16.1 Typedef Documentation

```
typedef struct pb_istream_s pb_istream_t
```

10.15.7.17 Typedef `pb_oiostream_t`

- Defined in [File nxp_iot_agent.h](#)

10.15.7.17.1 Typedef Documentation

```
typedef struct pb_oiostream_s pb_oiostream_t
```

10.15.7.18 Typedef pSe05xAppletFeatures_t

- Defined in [File se05x_tlv.h](#)

10.15.7.18.1 Typedef Documentation

```
typedef Se05x_AppletFeatures_t *pSe05xAppletFeatures_t
```

10.15.7.19 Typedef pSe05xPolicy_t

- Defined in [File se05x_tlv.h](#)

10.15.7.19.1 Typedef Documentation

```
typedef Se05xPolicy_t *pSe05xPolicy_t
```

10.15.7.20 Typedef pSe05xSession_t

- Defined in [File se05x_tlv.h](#)

10.15.7.20.1 Typedef Documentation

```
typedef Se05xSession_t *pSe05xSession_t
```

10.15.7.21 Typedef public_key_identifier_t

- Defined in [File nxp_iot_agent_service.h](#)

10.15.7.21.1 Typedef Documentation

```
typedef uint8_t public_key_identifier_t[16]
```

10.15.7.22 Typedef request_handler_callback_t

- Defined in [File nxp_iot_agent_dispatcher.h](#)

10.15.7.22.1 Typedef Documentation

```
typedef bool (*request_handler_callback_t)(pb_istream_t *istream, pb_ostream_t *ostream, const pb_field_t *message_type, void *endpoint_context)
```

10.15.7.23 Typedef SE05x_KeyID_t

- Defined in [File se05x_enums.h](#)

10.15.7.23.1 Typedef Documentation

```
typedef uint32_t SE05x_KeyID_t
```

SE05X's key IDs

10.15.7.24 Typedef SE05x_MacOperation_t

- Defined in [File se05x_enums.h](#)

10.15.7.24.1 Typedef Documentation

`typedef SE05x_MACAlgo_t SE05x_MacOperation_t`

HMAC/CMAC Algorithms

10.15.7.25 Typedef SE05x_MaxAttempts_t

- Defined in [File se05x_enums.h](#)

10.15.7.25.1 Typedef Documentation

`typedef uint16_t SE05x_MaxAttempts_t`

SE05X key's max attempts

10.15.7.26 Typedef SE05x_SecureObjectType_t

- Defined in [File se05x_enums.h](#)

10.15.7.26.1 Typedef Documentation

`typedef SE05x_SecObjTyp_t SE05x_SecureObjectType_t`

Type of Object

10.15.7.27 Typedef SE05x_Variant_t

- Defined in [File se05x_enums.h](#)

10.15.7.27.1 Typedef Documentation

`typedef SE05x_AppleConfig_t SE05x_Variant_t`

Features which are available / enabled in the Applet

10.15.7.28 Typedef sems_lite_PKGInfoList_t

- Defined in [File sems_lite_api.h](#)

10.15.7.28.1 Typedef Documentation

`typedef sems_lite_SEAppInfoList_t sems_lite_PKGInfoList_t`

Same as `sems_lite_SEAppInfoList_t` for list of installed packages

10.15.7.29 Typedef service_identifier_t

- Defined in [File nxp_iot_agent_service.h](#)

10.15.7.29.1 Typedef Documentation

```
typedef uint8_t service_identifier_t[32]
```

10.16 SE052 specific notes

10.16.1 Cmake options required for SE052

```
PTMW_Applet:STRING=SE05X_C  
PTMW_SCP:STRING=SCP03_SSS  
PTMW_SE05X_Auth:STRING=PlatfSCP03  
PTMW_SE05X_Ver:STRING=07_02  
PTMW_SE_RESET_LOGIC:STRING=1
```

To be applied on top of a configured host build area.

Also, update the OEF selection in feature file - fsl_sss_ftr.h (location - simw-top_build/<build_folder>),

```
#define SSS_PFSCP_ENABLE_SE052_B501 1
```

10.16.2 Runtime identification of SE051 vs SE052

During the session open with the secure element, the applet version is read / stored in session context to decide if a SE051 or SE052 is connected. This will ensure correct APDU commands are sent to secure element.

10.16.3 API Updates

- Updated SE05x API Se05x_API_GetFreeMemory
 - Argument pfreeMem is now uint32_t (earlier it was uint16_t)
 - All demos that refer to Se05x_API_GetFreeMemory are updated with this datatype change.
- Updated SE05x API Se05x_API_HKDF_Extended
 - External salt is mandatory. This is checked using new parameter applet_version. If applet_version is newer than 0x07020000, then we allow setting salt even if HKDF mode is ExpandOnly

10.16.4 Demo updates

se05x_GetInfo example

- With new JCOP, one extra TLV is returned in JCOPIdentify command. Added parsing of new TLV: JCOP OS Core ID
- Parsing is done based on new parameter applet_version. If applet_version is newer than 0x07020000, we parse the extra TLV, otherwise we continue parsing remaining data.

10.16.5 Self-Test trigger

Applet self-tests need to be triggered from the default session before using the respective algorithm: AES GCM/GMAC; PBKDF; TLS KDF,

10.16.6 APDU throughput Limitation

In case of throughput error – 0x66A6, invoke T=1I2C Chip reset command using API ‘phNxpEse_reset’ and reopen the session from the application. Refer [SE052 Reset Example](#)

11 Note about the source code in the document

Example code shown in this document has the following copyright and Apache-2.0 license: Copyright 2025 NXP Licensed under the Apache License, Version 2.0 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

12 Revision history

Document ID	Release date	Description
AN13030 v.2.5	22 July 2025	Corresponds to Plug and Trust middleware version v04_07_01. See section “2. Changes” for further details.
AN13030 v.2.4	6 March 2025	Corresponds to Plug and Trust middleware version v04_07_00. See section “2. Changes” for further details.
AN13030 v.2.3	28 March 2024	Corresponds to Plug and Trust middleware version v04_05_01. See section “2. Changes” for further details.
AN13030 v.2.2	21 December 2023	Corresponds to Plug and Trust middleware version v04_05_00. See section “2. Changes” for further details.
AN13030 v.2.1	14 March 2023	Corresponds to Plug and Trust middleware version v04_03_01. See section “2. Changes” for further details.
AN13030 v.2.0	12 January 2023	Corresponds to Plug and Trust middleware version v04_03_00. See section “2. Changes” for further details.
AN13030 v.1.9	30 June 2022	Corresponds to Plug and Trust middleware version v04_02_00. See section “2. Changes” for further details.
AN13030 v.1.6	15 December 2021	Corresponds to Plug and Trust middleware version v04_01_00. See section “2. Changes” for further details.
AN13030 v.1.5	27 October 2021	Corresponds to Plug and Trust middleware version v04_00_01. See section “2. Changes” for further details.
AN13030 v.1.4	1 July 2021	Corresponds to Plug and Trust middleware version v03_03_00. See section “2. Changes” for further details.

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at <https://www.nxp.com/profile/terms>, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Contents

1	NXP Plug & Trust Middleware	1	2.18.1	SEMSLite	12
1.1	Organization of Documentation	2	2.18.2	Other Miscellaneous Changes	12
1.2	Folder Structure	2	2.19	Release v04.01.00	13
1.3	List of Platform Prerequisites	3	2.19.1	SEMSLite	13
2	Changes	4	2.19.2	Examples / DEMO updates	13
2.1	Pending Refactoring items	4	2.19.3	Other Miscellaneous Changes	13
2.2	Known limitations	4	2.20	Release v04.00.01	13
2.3	Release v04.07.01	4	2.20.1	Build system changes	13
2.4	Release v04.07.00	5	2.20.2	Functional Changes	13
2.4.1	External module update	5	2.21	Release v04.00.00	13
2.4.2	New platform support	5	2.21.1	Build system changes	13
2.4.3	SSSCLI Changes	5	2.21.2	APIs & enum/types Changes	13
2.4.4	Functional Changes / New feature support	5	2.21.3	Functional Changes	14
2.4.5	SEMS Lite Change	7	2.21.4	New platform support	14
2.4.6	File/Folder relocation	7	2.21.5	SSSCLI Changes	15
2.4.7	Build system changes	7	2.21.6	Examples / DEMO updates	15
2.4.8	APIs & enum/types Changes	7	2.21.7	External modules Changes	15
2.4.9	EdgeLock 2GO Agent Changes	7	2.21.8	Other Miscellaneous Changes	15
2.5	Release v04.06.00	7	2.22	Release v03.03.00	15
2.6	Release v04.05.03	7	2.22.1	Build system changes	15
2.7	Release v04.05.02	8	2.22.2	Functional Changes	15
2.8	Release v04.05.01	8	2.22.3	New feature support	15
2.9	Release v04.05.00	8	2.22.4	SSSCLI Changes	15
2.9.1	Build system changes	8	2.22.5	Examples / DEMO updates	15
2.9.2	Functional Changes	9	2.22.6	Other Miscellaneous Changes	16
2.9.3	Examples / DEMO updates	9	2.23	Release v03.01.01	16
2.9.4	APIs & enum/types Changes	9	2.23.1	File/Folder relocation	16
2.10	Release v04.04.02	9	2.23.2	Build system changes	16
2.10.1	Changes	9	2.23.3	APIs & enum/types Changes	16
2.11	Release v04.04.01	10	2.23.4	Functional Changes	16
2.11.1	Changes	10	2.23.5	New platform support	16
2.12	Release v04.03.03	10	2.23.6	New feature support	16
2.12.1	Changes	10	2.23.7	SEMSLite	16
2.13	Release v04.03.02	10	2.23.8	SSSCLI Changes	16
2.13.1	Functional Updates	10	2.23.9	Documentation Changes	16
2.14	Release v04.03.01	10	2.23.10	Examples / DEMO updates	16
2.14.1	Important Security Updates	10	2.23.11	Communication Layer Changes	16
2.14.2	Other Miscellaneous Changes	10	2.23.12	EdgeLock 2GO agent	17
2.15	Release v04.03.00	11	2.23.13	User Interface Changes	17
2.15.1	Build system changes	11	2.23.14	External modules Changes	17
2.15.2	APIs & enum/types Changes	11	2.23.15	Other Miscellaneous Changes	17
2.15.3	Functional Changes	11	2.24	Release v03.01.00	17
2.15.4	New feature support	11	2.24.1	File/Folder relocation	17
2.15.5	SSSCLI Changes	11	2.24.2	Build system changes	17
2.15.6	Other Miscellaneous Changes	11	2.24.3	APIs & enum/types Changes	17
2.16	Release v04.02.00	11	2.24.4	Functional Changes	17
2.16.1	Build system changes	11	2.24.5	New platform support	18
2.16.2	APIs & enum/types Changes	11	2.24.6	New feature support	18
2.16.3	Functional Changes	12	2.24.7	SEMSLite	18
2.16.4	New feature support	12	2.24.8	SSSCLI Changes	18
2.16.5	SSSCLI Changes	12	2.24.9	Documentation Changes	18
2.16.6	Examples / DEMO updates	12	2.24.10	Examples / DEMO updates	18
2.16.7	Other Miscellaneous Changes	12	2.24.11	Communication Layer Changes	18
2.17	Release v04.01.03	12	2.24.12	EdgeLock 2GO agent	18
2.17.1	Other Miscellaneous Changes	12	2.24.13	User Interface Changes	18
2.18	Release v04.01.01	12	2.24.14	External modules Changes	19

2.24.15	Other Miscellaneous Changes	19	2.33.5	Functional Changes	27
2.25	Release v03.00.06	19	2.33.6	New feature support	27
2.25.1	Examples / DEMO updates	19	2.33.7	SSSCLI / PyCLI Changes	27
2.25.2	EdgeLock 2GO agent	19	2.33.8	Communication Layer Changes	27
2.26	Release v03.00.05	19	2.33.9	APIs & enum/types Changes	28
2.26.1	Functional Changes	19	2.33.10	Examples / DEMO updates	28
2.26.2	New feature support	19	2.33.11	Documentation Changes	28
2.26.3	SSSCLI Changes	19	2.33.12	Other Miscellaneous Changes	28
2.26.4	Documentation Changes	19	2.34	Release v02.12.05	29
2.26.5	Examples / DEMO updates	19	2.35	Release v02.12.04	29
2.27	Release v03.00.04	20	2.36	Release v02.12.03	29
2.27.1	Build system changes	20	2.37	Release v02.12.02	29
2.27.2	SSSCLI / PyCLI Changes	20	2.37.1	AKM Specific changes	29
2.27.3	Documentation Changes	20	2.37.2	Example Updates	29
2.27.4	Examples / DEMO updates	20	2.37.3	Other Miscellaneous Changes	29
2.27.5	Other Miscellaneous Changes	20	2.38	Release v02.12.01	29
2.28	Release v03.00.03	20	2.38.1	AKM Specific changes	29
2.28.1	Build system changes	20	2.38.2	New example / example updates	29
2.28.2	APIs & enum/types Changes	20	2.38.3	Other Miscellaneous Changes	29
2.28.3	New platform support	21	2.39	Release v02.12.00	30
2.28.4	New feature support	21	2.39.1	New feature support	30
2.28.5	SEMSLite	21	2.39.2	SSSCLI / PyCLI Changes	30
2.28.6	Examples / DEMO updates	21	2.39.3	Documentation Changes	30
2.28.7	Communication Layer Changes	21	2.40	Release v02.11.03	30
2.28.8	External modules Changes	21	2.40.1	File/Folder relocation	30
2.28.9	Other Miscellaneous Changes	21	2.40.2	New feature support	30
2.29	Release v03.00.02	22	2.40.3	Scripts and Build changes	30
2.29.1	Build system changes	22	2.40.4	Documentation Changes	31
2.29.2	APIs & enum/types Changes	22	2.40.5	Other Miscellaneous Changes	31
2.29.3	Functional Changes	22	2.41	Internal Release v02.11.01	31
2.29.4	New feature support	22	2.41.1	APIs & enum/types Changes	31
2.29.5	SSSCLI / PyCLI Changes	22	2.41.2	Functional Changes	31
2.29.6	Examples / DEMO updates	22	2.41.3	New feature support	31
2.29.7	Other Miscellaneous Changes	22	2.41.4	Scripts and Build changes	32
2.30	Release v02.16.01	23	2.41.5	Documentation Changes	32
2.30.1	APIs & enum/types Changes	23	2.41.6	Communication Layer Changes	32
2.30.2	Documentation Changes	23	2.41.7	User Interface Changes	32
2.30.3	Other Miscellaneous Changes	23	2.41.8	Other Miscellaneous Changes	32
2.31	Release v02.16.00	23	2.42	Release v02.11.00	32
2.31.1	File/Folder relocation	23	2.42.1	File/Folder relocation	32
2.31.2	New platform support	24	2.42.2	APIs & enum/types Changes	34
2.31.3	New feature support	24	2.42.3	Functional Changes	34
2.31.4	SSSCLI / PyCLI Changes	24	2.42.4	New platform support	35
2.31.5	Examples / DEMO updates	24	2.42.5	Scripts and Build changes	35
2.31.6	Communication Layer Changes	24	2.42.6	SSSCLI / PyCLI Changes	35
2.31.7	Other Miscellaneous Changes	24	2.42.7	Documentation Changes	35
2.32	Release v02.15.00	24	2.42.8	Communication Layer Changes	35
2.32.1	Build system changes	24	2.42.9	Other Miscellaneous Changes	36
2.32.2	APIs & enum/types Changes	25	2.43	Release v02.10.00	36
2.32.3	Functional Changes	25	2.43.1	APIs & enum/types Changes	36
2.32.4	New feature support	25	2.43.2	New platform support	37
2.32.5	Documentation Changes	25	2.43.3	Scripts and Build changes	37
2.32.6	Examples / DEMO updates	25	2.43.4	SSSCLI / PyCLI Changes	37
2.32.7	Other Miscellaneous Changes	25	2.43.5	Other Miscellaneous Changes	37
2.33	Release v02.14.00	26	2.44	Release v02.09.00	37
2.33.1	File/Folder relocation	26	2.44.1	Middleware	37
2.33.2	Build system changes	26	2.44.2	CLI Tool	37
2.33.3	New platform support	26	2.45	Release v02.07.00	38
2.33.4	APIs & enum/types Changes	26	2.45.1	Middleware	38

2.45.2	SMCom	38	3.12	Auth Objects : AESKey	154
2.45.3	SSS APIs	38	3.13	Auth Objects : ECKey	154
2.45.4	CLI Tool	38	3.13.1	ECKey - Keys Used	155
2.45.5	Development Platform	38	3.13.2	ECKey - Use for connection / authentication ..	155
2.46	Release v02.06.00	38	3.14	Key Id Range and Purpose	156
2.46.1	Plug & Trust Middleware	38	3.15	Authentication Keys	157
2.46.2	Documentation	39	3.16	Trust provisioned KeyIDs	157
2.46.3	Development Platform	39	3.17	SCP03 with PUF	157
2.47	Release v02.05.00	39	3.17.1	Activation Code	158
2.48	Release v02.04.00	39	3.17.2	Key Code	158
2.49	Release 02.03.00	39	3.17.3	Using with LPC55S	158
3	Plug & Trust MW Stack	39	3.18	SSS Heap Management	158
3.1	Features	39	3.19	Secure Boot	159
3.2	Plug & Trust MW : Block Diagram	40	3.19.1	Secure Boot flow	160
3.3	SSS APIs	40	3.19.2	Secondary Bootloader	160
3.3.1	SSS: Introduction	40	3.19.3	Secure Binding	160
3.3.2	Session	40	4	Building / Compiling	162
3.3.3	Key Store	41	4.1	Windows Build	163
3.3.4	Key Object	42	4.1.1	Prerequisite	163
3.3.5	Asymmetric	46	4.1.2	Create Build files	163
3.3.6	Policies	51	4.1.3	SSS Examples	163
3.3.7	Example Boot-Up	54	4.2	Import MCUXpresso projects from SDK	163
3.3.8	SSS api key format (asymmetric keys)	56	4.2.1	Prerequisite	163
3.3.9	List of all SSS APIs and structures	64	4.2.2	Importing an example	164
3.3.10	List of all SSS SE05x APIs and structures	103	4.2.3	Running / Debugging example	166
3.4	SSS APIs: SE051 vs SE050	135	4.2.4	Logging on console	169
3.5	Parameter Check & Conventions	137	4.3	Freedom K64F (and Other MCUs) Build (CMake - Advanced)	169
3.5.1	Parameter Convention	137	4.3.1	Prerequisite	170
3.5.2	Helper Macros	138	4.3.2	Importing the Project	170
3.5.3	Apis	138	4.3.3	Running / Debugging example	172
3.6	I2CM / Secure Sensor	140	4.3.4	Logging on console	175
3.6.1	Normal Read/Write	140	4.4	i.MX Linux Build	175
3.6.2	Attested Read	141	4.4.1	Prerequisite	175
3.6.3	Transaction	142	4.4.2	Build Instructions	175
3.6.4	Read with Attestation	142	4.4.3	SSS Examples	177
3.6.5	I2C Master APIs	142	4.4.4	Raspberry Pi Build	177
3.7	Logging	142	4.5	Prerequisite	177
3.7.1	Logging level	143	4.5.1	Connecting SE05X with RaspberryPi	177
3.7.2	Adding log messages into the source code ..	143	4.5.2	Enable Pin configuration for SE05X	177
3.7.3	Logging APIs	143	4.5.3	Build Instructions	177
3.7.4	Logging Header Files	146	4.5.4	SSS Examples	179
3.7.5	Changing logging level	146	4.5.5	CMake	179
3.8	Feature File - fsl_sss_ftr.h	148	4.6	Reference Commands	179
3.8.1	When Using CMake	148	4.6.1	CMake - Cross compiling	179
3.8.2	When Using MCUXpresso IDE	148	4.6.2	CMake Options	180
3.8.3	fsl_sss_ftr_default.h	148	4.7	PTMW_Applet	180
3.8.4	Using feature file to reduce code size	149	4.7.1	PTMW_SE05X_Ver	180
3.8.5	SSS_HAVE_APPLET_SE05X_A	149	4.7.2	PTMW_Host	180
3.8.6	SSS_HAVE_APPLET_SE05X_B	149	4.7.3	PTMW_SMCOM	181
3.8.7	SSS_HAVE_APPLET_SE05X_C	149	4.7.4	PTMW_HostCrypto	181
3.9	Platform SCP03 Authentication	151	4.7.5	PTMW_RTOS	181
3.9.1	MW Configuration	151	4.7.6	PTMW_mbedTLS_ALT	182
3.9.2	Using Platform SCP Keys from File System ..	151	4.7.7	PTMW_SCP	182
3.10	Auth Objects	152	4.7.8	PTMW_FIPS	182
3.11	Auth Objects : UserID	152	4.7.9	PTMW_SBL	182
3.11.1	User ID - Provisioning / Injection	152	4.7.10	PTMW_SE05X_Auth	182
3.11.2	User ID - Use for connection / authentication	153	4.7.11	PTMW_Log	183
3.11.3	User ID - Applet Spec Notes	154	4.7.12	CMAKE_BUILD_TYPE	183

4.7.14	PTMW_OpenSSL	183	5.6.2	Preparing the credentials and Provisioning the secure element	208
4.7.15	PTMW_MBedTLS	183	5.6.3	Registering Device	209
4.7.16	PTMW_SE_RESET_LOGIC	183	5.6.4	Create device enrollment in azure IoT Hub portal	209
4.7.17	Feature Control	184	5.6.5	Uploading root certificates to IoT Hub	210
4.7.18	Deprecated Defines	185	5.6.6	Build the OpenSSL engine [Optional]	210
4.7.19	NXP Internal Options	186	5.6.7	Run the example	211
4.7.20	Other Variables	186	5.7	OpenSSL Engine / Provider: TLS Client example for iMX/Rpi3	212
5	Demo and Examples	187	5.7.1	Summary	212
5.1	DEMO List	187	5.7.2	Credential preparation (execute once) [Optional]	212
5.1.1	Platforms List	187	5.7.3	Secure Element preparation (client side)	213
5.1.2	SSS APIs Examples	187	5.7.4	Server side preparation	214
5.1.3	Cloud connectivity Examples	188	5.7.5	Start up the server	214
5.1.4	OpenSSL Engine Examples	188	5.7.6	Establish a TLS link from the client to the server	214
5.1.5	mbedTLS Examples	188	5.8	Access Manager: Manage access from multiple (Linux) processes to an SE05x IoT Applet	215
5.1.6	OPC UA Examples	189	5.8.1	Summary	215
5.1.7	PSA / TF-M Examples	189	5.8.2	Usage	216
5.1.8	SE05X Specific Examples	189	5.8.3	Build	216
5.1.9	Examples that use OpenSSL	191	5.8.4	Demo: concurrent access from 2 processes using OpenSSL engine	217
5.1.10	Ease of Use Configuration Examples	191	5.8.5	Example programs prepared for concurrent access	217
5.1.11	SEMS Demos	191	5.8.6	Concepts & Features	218
5.1.12	LPC55S-PUF Based examples	191	5.8.7	Restrictions	219
5.1.13	EdgeLock 2GO Agent example	191	5.8.8	Session open retry in client processes (Only for SE05X)	220
5.2	SSS API Examples	192	5.8.9	Access Manager with Unix sockets	220
5.2.1	ECC Example	192	5.8.10	Access Manager command to reset secure element	220
5.2.2	RSA Example	193	5.9	OPC UA (Open62541) Demo	221
5.2.3	Symmetric AES Example	193	5.9.1	Supported Platforms	221
5.2.4	HKDF Example	194	5.9.2	Introduction	221
5.2.5	Message Digest Example	195	5.9.3	Build Open62541 server and client examples	222
5.2.6	HMAC Example	196	5.9.4	Test Open62541 server and client examples	222
5.2.7	ECDH Example	196	5.9.5	Known Limitations	223
5.2.8	ECC NIST256 Key Attestation Example	197	5.10	PSA Non Secure Example	223
5.2.9	ECC MONTGOMERY-25519 Key Attestation Example	198	5.10.1	Pre-requisites	223
5.2.10	EDDSA Example	199	5.10.2	PSA Operation Examples	224
5.3	AWS Demo for MCU-SDK	200	5.10.3	Building Example	224
5.3.1	Prerequisites	200	5.11	SE05X Minimal example	225
5.3.2	Using WiFi with LPC55S	201	5.11.1	Building the Demo	225
5.3.3	Creating a device on AWS account	201	5.11.2	Running the Example	225
5.3.4	Creating and updating device keys and certificates to SE	201	5.11.3	Console output	225
5.3.5	Building the Demo	201	5.12	SE05X Get Info example	225
5.3.6	Running the Demo	202	5.12.1	Building the Demo	225
5.4	AWS Demo for iMX Linux / RaspberryPi	202	5.12.2	Running the Example	225
5.4.1	Prerequisites	202	5.12.3	Console output	226
5.4.2	Preparing the credentials and Provisioning the secure element	203	5.13	APDU Player Demo	227
5.4.3	Run the example	204	5.13.1	How to use	228
5.5	Azure Demo for MCU-SDK	205	5.14	Using policies for secure objects	228
5.5.1	Prerequisites	206	5.14.1	Sign Policy	228
5.5.2	Using WiFi with LPC55S	206			
5.5.3	Creating a device on azure IoT Hub portal	206			
5.5.4	Creating and updating device keys and certificates to SE	206			
5.5.5	Uploading root certificates to IoT Hub	206			
5.5.6	Building the Demo	207			
5.5.7	Running the Demo	207			
5.5.8	Appendix	208			
5.6	Azure Demo for iMX Linux / Raspberry Pi	208			
5.6.1	Prerequisites	208			

5.14.2	Using PCR Object	230	5.31	SE05X Personalization Remove RSA Key Generation Module	257
5.14.3	Console output	231	5.31.1	Building the Demo	258
5.15	Get Certificate from the SE	231	5.31.2	Running the Example	258
5.15.1	Building the example	231	5.31.3	Console output	258
5.15.2	How to use	231	5.32	DEMO for Personalization of SE051	258
5.16	SE05X Rotate PlatformSCP Keys Demo	232	5.32.1	Compatible settings from host and SE	259
5.16.1	Prerequisites	232	5.33	SE05X MultiThread demo	259
5.16.2	Configuring the Demo	232	5.33.1	Prerequisites	259
5.16.3	Building the Demo	234	5.33.2	Building the Demo	259
5.16.4	Running the Example	234	5.33.3	Running the Example	259
5.16.5	Console output	234	5.33.4	Console output	260
5.17	I2C Master Example	234	5.34	SE05X Invoke Garbage Collection Example	260
5.17.1	Prerequisites	234	5.34.1	Building the Demo	260
5.17.2	About the Example	237	5.34.2	Running the Example	260
5.17.3	Running the Demo	238	5.34.3	Console output	261
5.18	SE05X WiFi KDF Example	238	5.35	ECC Concurrent Example	261
5.18.1	Building the Demo	238	5.35.1	Building the Demo	261
5.18.2	Running the Example	238	5.35.2	Restrictions	261
5.18.3	Console output	239	5.35.3	Running the Example in combination with the Access Manager	261
5.19	SE05X Export Transient objects	239	5.35.4	Usage	261
5.19.1	Visual Studio - Debug settings	239	5.35.5	Concurrent Usage	262
5.19.2	Console output	239	5.36	Symmetric Multi Step Concurrent Example	262
5.20	SE05X Import Transient objects	240	5.36.1	Building the Demo	262
5.20.1	Visual Studio - Debug settings	240	5.36.2	Restrictions	262
5.20.2	Console output	240	5.36.3	Running the Example in combination with the Access Manager	263
5.21	Import External Object Prepare	240	5.36.4	Usage	263
5.21.1	Building	245	5.36.5	Concurrent Usage	263
5.21.2	How to use	245	5.37	Delete and Test Provision	264
5.22	SE05X Mandate SCP example	245	5.37.1	Object IDs provisioned	264
5.22.1	Building the Demo	245	5.37.2	Authentication Keys	265
5.22.2	Running the Example	245	5.37.3	Building the Demo	266
5.23	Read object with Attestation	246	5.38	se05x Multiple Digest Crypto Objects example	266
5.23.1	Building	246	5.38.1	Building the Demo	266
5.23.2	Running	246	5.38.2	Running the Example	266
5.23.3	Reading large binary objects with attestation	247	5.38.3	Console output	267
5.24	SE05X Transport Lock example	251	5.39	SE05X Allow Without SCP example	267
5.25	SE05X Transport UnLock example	251	5.39.1	Building the Demo	267
5.26	SE05X Timestamp	251	5.39.2	Running the Example	267
5.26.1	Building	252	5.40	SE05X SCP03 BOOT Example	268
5.26.2	Running	252	5.40.1	Prerequisites	268
5.27	SE05X PCR example	252	5.40.2	Running the Example	268
5.27.1	Building the Demo	252	5.41	Secure Authenticator (Qi) Authentication demo	268
5.27.2	Running the Example	252	5.41.1	Pre-requisites	269
5.27.3	Console output	252	5.41.2	GetCertificateChainDigest (GET_DIGESTS)	269
5.28	Configuring Applet Features	255	5.41.3	ReadCertificates (GET_CERTIFICATE)	272
5.28.1	Configuring ECKey key for command line	255	5.41.4	Authenticate (CHALLENGE)	274
5.28.2	Configuring Environment for PlatfSCP03 Keys	255	5.41.5	Building the Demo	276
5.28.3	Using with PC as host	255	5.41.6	Running the Example	277
5.28.4	Using with iMX-6 as host	256	5.41.7	Porting	278
5.29	Inject Certificate into SE	256	5.42	SE052 Reset Example	279
5.29.1	Building the example	256	5.42.1	Building the Demo	279
5.29.2	How to use	256	5.42.2	Running the Example	279
5.30	SE05X Read State example	256	5.43	SE052 Deep Power down example	279
5.30.1	Building the Demo	257			
5.30.2	Running the Example	257			
5.30.3	Console output	257			

5.43.1	Building the Demo	280	6.2	Building and running the EdgeLock 2GO agent	301
5.43.2	Running the Example	280	6.2.1	Building / Compiling the EdgeLock 2GO agent	301
5.44	SE05X Multi Thread and Multi session example	280	6.2.2	Registering the device to the EdgeLock 2GO service	302
5.44.1	Building the Demo	280	6.2.3	Connecting the device to the EdgeLock 2GO service	302
5.44.2	Running the Example	280	6.2.4	OpenSSL engine/provider configuration	302
5.45	SE05X Access Manager Reset example	280	6.2.5	Re-provisioning of objects	303
5.45.1	Building the Demo	281	6.3	Datastore / Keystore	303
5.45.2	Running the Example	281	6.4	Connection to the EdgeLock 2GO cloud service	303
5.46	Tool to create Reference key file	281	6.4.1	Transport layer security	303
5.46.1	Building the example	281	6.4.2	Client authentication	304
5.46.2	How to use	281	6.4.3	Server authentication	304
5.47	Building a self-signed certificate	282	6.4.4	Application layer protocol	304
5.47.1	How to use	282	6.4.5	Parameters for the connection to EdgeLock 2GO cloud service	305
5.48	Tests for User Crypto	282	6.5	Claim Codes	305
5.48.1	List of Crypto Algorithms	283	6.6	Offline Provisioning of Secure Objects	305
5.48.2	Building the Demo	283	6.6.1	Offline Remote Trust Provisioning Server (RTP Server)	306
5.49	Ease of Use configuration - Azure IoT Hub	283	6.6.2	Offline Remote Trust Provisioning Client (RTP Client)	306
5.49.1	Creating Device on Azure DPS	283	6.7	API	307
5.49.2	Registering Device to IoT Hub	284	6.7.1	EdgeLock 2GO Main	307
5.49.3	Running Azure Demo	284	6.7.2	EdgeLock 2GO Datastore	313
5.49.4	Update cloud example	284	6.7.3	EdgeLock 2GO Keystore	315
5.49.5	Build and run the demo	285	6.7.4	EdgeLock 2GO Session	316
5.50	Ease of Use configuration - AWS IoT Console	285	6.7.5	EdgeLock 2GO Service	316
5.50.1	Pre-requisites	285	6.7.6	EdgeLock 2GO Dispatcher	319
5.50.2	Extracting Device Certificate	285	6.7.7	EdgeLock 2GO Utils	320
5.50.3	Registering Device Certificate	285	6.8	EdgeLock 2GO Agent Examples	326
5.50.4	Running on Linux	286	6.8.1	Initializaton of context and updating device configuration	326
5.50.5	Running on MCU	287	6.8.2	MQTT connection tests with Cloud Onboarding Provisioning	328
5.51	SEMS Lite Agent Demo (sems_lite_ex_update)	287	6.8.3	MQTT connection tests with Remote Trust Provisioning	332
5.51.1	Prerequisites	287	7	SEMS Lite Agent	337
5.51.2	Running the Example	287	7.1	SEMS Lite Overview (Only for SE051)	337
5.52	SEMS Lite CLI APP	287	7.1.1	Update Manager (from customer)	337
5.52.1	Building the CLI APP	288	7.1.2	SEMS Lite Agent (from NXP)	337
5.52.2	CLI Parameters	288	7.2	Update Package	338
5.52.3	Usage - K64F	289	7.2.1	Package Meta Data	338
5.52.4	Usage - Raspberry Linux	289	7.2.2	MulticastCommands In Protocol Buffer Format	339
5.52.5	Version Compatibility	289	7.2.3	Generator Tools	340
5.53	Key Injection to PUF	289	7.3	SEMS Lite Agent Usage	343
5.53.1	Pre-requisites	289	7.3.1	Case 1: Basic Usage	343
5.53.2	How to build	289	7.3.2	Case 2: In case there was a tearing	343
5.53.3	How to use	290	7.3.3	Case 3: Doing a recovery after a failed update	345
5.53.4	Injecting keys into PUF	290	7.3.4	Case 4: Tearing during recovery	348
5.54	Key Rotation using PUF	291	7.3.5	Case 5: Load Key rotate script encounter	349
5.54.1	Pre-requisites	291	7.3.6	SE unexpected power off	349
5.54.2	How to build	291	7.4	API Calls for Usage of SEMS Lite Agent	350
5.54.3	How to run	292		SEMS Lite management APIs	351
5.55	Secure Boot Demo	295			
5.55.1	Secondary Bootloader (SBL)	295			
5.55.2	Secure Application	295			
5.55.3	Non-Secure Application	295			
5.55.4	Prerequisites	295			
5.55.5	How to build	296			
5.55.6	How to run	299			
6	NXP EdgeLock 2GO Agent	301			
6.1	Introduction	301			

7.4.1	Get UUID	351	8.9.1	PKCS#11 Label Handling	395
7.4.2	Get Public Key	352	8.9.2	PKCS#11 Object caching	395
7.4.3	Get Card contents	352	8.9.3	PKCS#11 specifications	395
7.5	SEMS Lite Agent Package Load Process	353	8.9.4	Building on Linux / Raspberry-Pi	397
7.6	SEMSLite Types and APIs	355	8.9.5	Using with pkcs11-tool	398
7.6.1	SEMSLite Agent Types	355	8.9.6	Notes	398
7.6.2	SEMSLite Agent APIs	360	9	CLI Tool	398
7.7	SEMS Lite Known Issue	367	9.1	Introduction	398
7.7.1	Signature for rerun script is incorrect occasionally	367	9.2	Block Diagram	399
7.8	SEMS Lite DEMOs	367	9.3	Steps needed before running ssscli tool	399
8	Plugins / Add-ins	367	9.3.1	Once per installation	399
8.1	Introduction on OpenSSL engine	367	9.3.2	Communication interface (cmake SMCOM setting)	401
8.1.1	General	368	9.4	Running the ssscli tool - Windows	401
8.1.2	Keys	368	9.5	CLI Provisioning	401
8.1.3	Building the OpenSSL engine	370	9.5.1	Generating keys and certificates	401
8.1.4	Sample scripts to demo OpenSSL Engine	370	9.5.2	Provisioning for the demo	401
8.2	Introduction on OpenSSL provider	378	9.5.3	Steps to provision your device for demo on Windows	401
8.2.1	OpenSSL versions	379	9.5.4	Steps to provision your device for demo on iMX or Raspberry Pi	402
8.2.2	Platforms	379	9.6	Usage Examples	403
8.2.3	Key formats supported	379	9.6.1	SE05X: VCOM Interface	403
8.2.4	Building the OpenSSL provider	379	9.6.2	SE05X: PCSC interface	405
8.2.5	Sample scripts to demo OpenSSL provider	380	9.6.3	se05x: JRCPV2 interface	406
8.2.6	OSSL Algorithms property definitions	380	9.6.4	MBEDTLS	408
8.3	Introduction on MbedTLS ALT (2.x) Implementation	380	9.7	List of ssscli commands	409
8.3.1	SE05X usage in TLS handshake (ECC Cipher suites)	380	9.7.1	ssscli Commands	409
8.3.2	SE05X usage in TLS handshake (RSA Cipher suites)	381	9.7.2	Set Commands	413
8.3.3	Using SE05X for all Public key ECDSA verify operation	381	9.7.3	Get Commands	415
8.3.4	Using mbedTLS ALT	381	9.7.4	Generate Commands	417
8.3.5	Testing on x86 / Linux	383	9.7.5	Refpem Commands	417
8.3.6	Testing mbedTLS client on frdm-k64	384	9.7.6	Se05x Commands	418
8.3.7	SE050 Performance Measurements	384	9.7.7	POLICY Commands	419
8.3.8	mbedTLS ALT APIs	385	9.8	CLI Data formats	423
8.4	Introduction on Mbed TLS (3.x) ALT Implementation	386	9.8.1	DER	423
8.4.1	Key Management	386	9.8.2	PEM	423
8.4.2	Build / Configuration of Mbed TLS ALT files	386	9.8.3	HEX	423
8.4.3	SE05X usage in TLS handshake	387	9.8.4	REFERENCE KEY	424
8.4.4	Testing Mbed TLS ALT (Windows)	387	9.9	Object Policies Through ssscli	424
8.5	Platform Security Architecture	388	9.9.1	Create object policy	424
8.5.1	PSA SE Driver Interface	388	9.9.2	Attach policy to object	427
8.5.2	PSA Concepts	389	9.10	Upload keys and certificates to SE05X using ssscli tool	428
8.5.3	Managing KeyIDs	389	10	Appendix	428
8.5.4	Building PSA for TrustZone	390	10.1	Glossary	428
8.6	Android Key master	390	10.2	VCOM	429
8.7	Introduction on Open62541 (OPC UA stack)	390	10.2.1	Steps to use VCOM for running the examples - PC & FRDM-K64F	429
8.7.1	Integrating SE050 in Open62541	390	10.3	APDU Commands over VCOM	430
8.8	WiFi EAP Demo with Raspberry Pi3	391	10.3.1	COM port parameters	430
8.8.1	Prerequisites	391	10.3.2	VCOM Format	430
8.8.2	Introduction	391	10.3.3	Example Commands	430
8.8.3	Setting up Access point	391	10.4	Visual Studio 2019 Setup	432
8.8.4	Setting up freeradius Server on Ubuntu	391	10.4.1	Prerequisites	432
8.8.5	Setting up Raspberry Pi3	393	10.4.2	Installing the components	432
8.9	PKCS#11 Standalone Library	395	10.5	Setting up MCUXpresso IDE	434
			10.5.1	To Download and install MCUXpresso IDE: ...	434

10.5.2	To Install board specific SDK in MCUXpresso	435
10.6	Development Platforms	435
10.6.1	Setup i.MX 8MQuad - MCIMX8M-EVK	435
10.6.2	Setup i.MX6UL - MCIMX6UL-EVK	440
10.6.3	Freedom K64F	444
10.6.4	i.MX RT 1060	444
10.6.5	LPC55S69	444
10.6.6	MIMXRT1170	446
10.6.7	FRDM-MCXA153	447
10.6.8	FRDM-MCXN947	447
10.6.9	Android: Hikey960	448
10.7	How to get SE Platform Information and UID	460
10.7.1	Using TeraTerm and pre-built binary	460
10.7.2	Using VCOM and binary	460
10.7.3	Using ssscli Tool	461
10.7.4	SE Platform Information on Android platform	461
10.8	Version Information	462
10.9	Certificate Chains	463
10.9.1	SE050 Certificate Chains	463
10.9.2	SE051 Certificate Chains	466
10.10	JRCP_v1 Server	470
10.10.1	Introduction	470
10.10.2	Using the JRCP_v1 server	471
10.11	Using own Platform SCP03 Keys	471
10.12	Write APDU to buffer	472
10.13	Class Hierarchy	472
10.14	File Hierarchy	476
10.15	Full API	477
10.15.1	Classes and Structs	477
10.15.2	Enums	515
10.15.3	Unions	562
10.15.4	Functions	563
10.15.5	Variables	769
10.15.6	Defines	769
10.15.7	Typedefs	803
10.16	SE052 specific notes	808
10.16.1	Cmake options required for SE052	808
10.16.2	Runtime identification of SE051 vs SE052	808
10.16.3	API Updates	808
10.16.4	Demo updates	808
10.16.5	Self-Test trigger	808
10.16.6	APDU throughput Limitation	808
11	Note about the source code in the document	809
12	Revision history	809
	Legal information	810

Please be aware that important notices concerning this document and the product(s) described herein, have been included in section 'Legal information'.