SE\_STSAFEA\_API

V1.0.0

Contents

[1. Introduction 5](#_Toc85469378)

[2. Principles 5](#_Toc85469379)

[2.1. STSAFE-A Secure Channel 6](#_Toc85469380)

[2.2. STSAFE-A Pairing Keys 6](#_Toc85469381)

[2.3. STSAFE-A Auto management 7](#_Toc85469382)

[2.4. I2C configuration 7](#_Toc85469383)

[2.5. CMake configuration 8](#_Toc85469384)

[2.6. Linux specific behavior 8](#_Toc85469385)

[3. Crypto Library 9](#_Toc85469386)

[3.1. MbedTLS 9](#_Toc85469387)

[3.2. OpenSSL 9](#_Toc85469388)

[4. API description 10](#_Toc85469389)

[4.1. Flags and Structures descriptions 10](#_Toc85469390)

[4.1.1. Flags 10](#_Toc85469391)

[4.1.2. Error Codes 10](#_Toc85469392)

[4.1.3. Structures 10](#_Toc85469393)

[4.1.3.1. SE\_Infot\_t 10](#_Toc85469394)

[4.1.3.2. SE\_Product\_Data\_t 11](#_Toc85469395)

[4.1.3.3. SE\_I2C\_Params\_t 11](#_Toc85469396)

[4.1.3.4. SE\_Pairing\_Info\_t 11](#_Toc85469397)

[4.1.3.5. SE\_Envelop\_Info\_t 11](#_Toc85469398)

[4.1.3.6. SE\_State\_t 11](#_Toc85469399)

[4.1.3.7. SE\_Cmd\_t 12](#_Toc85469400)

[4.1.3.8. SE\_Region\_t 12](#_Toc85469401)

[4.1.3.9. SE\_SerialFormat\_t 12](#_Toc85469402)

[4.1.3.10. SE\_Slot\_t 12](#_Toc85469403)

[4.1.3.11. SE\_Hash\_Type\_t 12](#_Toc85469404)

[4.1.3.12. SE\_Key\_Type\_t 13](#_Toc85469405)

[4.1.3.13. SE\_Security\_Level\_t 13](#_Toc85469406)

[4.1.3.14. SE\_Wrap\_Type\_t 13](#_Toc85469407)

[4.2. Init and status commands 14](#_Toc85469408)

[4.2.1. SE\_Init 14](#_Toc85469409)

[4.2.2. SE\_SetSecurity 14](#_Toc85469410)

[4.2.3. SE\_GetSerial 14](#_Toc85469411)

[4.2.4. SE\_GetKeySize 14](#_Toc85469412)

[4.2.5. SE\_GetCommandStatus 15](#_Toc85469413)

[4.2.6. SE\_GetMapping 15](#_Toc85469414)

[4.2.7. SE\_GetRandom 15](#_Toc85469415)

[4.2.8. SE\_GetInfo 15](#_Toc85469416)

[4.3. Management command 16](#_Toc85469417)

[4.3.1. SE\_Reset 16](#_Toc85469418)

[4.3.2. SE\_Hibernate 16](#_Toc85469419)

[4.3.3. SE\_Echo 16](#_Toc85469420)

[4.3.4. SE\_Refresh 16](#_Toc85469421)

[4.3.5. SE\_Raw 17](#_Toc85469422)

[4.4. Enablement commands 18](#_Toc85469423)

[4.4.1. SE\_GetState 18](#_Toc85469424)

[4.4.2. SE\_Unlock 18](#_Toc85469425)

[4.4.3. SE\_Pairing 18](#_Toc85469426)

[4.4.4. SE\_GetPairingStatus 18](#_Toc85469427)

[4.4.5. SE\_SetPairingKeys 19](#_Toc85469428)

[4.5. Memory partition access 20](#_Toc85469429)

[4.5.1. SE\_Read 20](#_Toc85469430)

[4.5.2. SE\_ReadRight 20](#_Toc85469431)

[4.5.3. SE\_Update 20](#_Toc85469432)

[4.5.4. SE\_UpdateRight 21](#_Toc85469433)

[4.6. Local Secure Storage commands 22](#_Toc85469434)

[4.6.1. Se\_GenerateWrapKey 22](#_Toc85469435)

[4.6.2. SE\_Wrap 22](#_Toc85469436)

[4.6.3. SE\_Unwrap 22](#_Toc85469437)

[4.7. Asymmetric primitive 23](#_Toc85469438)

[4.7.1. SE\_VerifySignature 23](#_Toc85469439)

[4.7.2. SE\_GenerateSignature 23](#_Toc85469440)

[4.7.3. SE\_GenerateKeyPair 23](#_Toc85469441)

[4.7.4. SE\_DeriveSharedSecret 24](#_Toc85469442)

[4.8. Application level Key and certificate management 25](#_Toc85469443)

[4.8.1. SE\_StoreCertificate 25](#_Toc85469444)

[4.8.2. SE\_GetCertificate 25](#_Toc85469445)

[4.8.3. SE\_SetActiveKey 25](#_Toc85469446)

[4.8.4. SE\_GetActiveKey 25](#_Toc85469447)

[4.8.5. SE\_GetActiveCertificate 25](#_Toc85469448)

[4.8.6. SE\_SetActiveCertificate 26](#_Toc85469449)

[5. Examples Description 27](#_Toc85469450)

[5.1. stsafea\_describe 27](#_Toc85469451)

# Introduction

SE\_STSAFEA\_API propose several components to add on top of X-CUBE-SAFEA1 package to manage in a more comprehensive way STSAFE-A110 feature

SE\_STSAFEA\_API provides abstraction to X-CUBE-SAFEA1 API and extends its function.

SE\_STSAFEA\_API also provides API to manage the STSAFE-A enablement procedure and extra API to manage Key and certificate.

# Principles

Crypto API

Crypto API

SE\_STSAFEA\_API

Secure Storage

X\_CUBE\_SAFEA1

I2C Interface

Crypto Interface

Certificat management

Asymmetric Crypto

SE\_STSAFEA\_API will manage the STSAFE-A handle and secure channel.

The Init function will gather extra information regarding key size and can execute basic echo test and pairing test.

## STSAFE-A Secure Channel

STSAFE-A Secure Channel is composed on 2 mechanisms:

* Command authorization (Command MAC)
* Payload data encryption

The Payload data encryption apply only to few commands set and is configurable at personalization.

The SE\_STSAFEA\_API will automatically get the STSAFE-A configuration to apply configuration to the relevant commands.

For instance, the StSafeA\_EstablishKey() does not mandate secure channel with evaluation samples SPL02 (STSAFA110xxSPL02 or X-NUCLEO-SAFEA1) but it is generally activated on personalized samples (check with your FAE for the configuration)

To handle seamlessly the configuration, the SE\_STSAFEA\_API is getting the default configuration of your STSAFE-A at the Init.

The command authorization is optional except for:

* Region access (depending on the region configuration)
* Specific command set configured at personalization.

For region access, the SE\_STSAFEA\_API is getting the memory mapping and configuration to automatically activate the command authorization if needed.

Nevertheless, the user can request Response verification MAC (R-MAC)

The user can activate the Response verification MAC by using the SE\_SetSecurityLevel() with SE\_SECURITY\_LEVEL\_R\_PAIRING.

Therefore, per commands, the SE\_STSAFEA\_API will check if the command need specific authorization, then check the user security level to use the appropriate secure channel.

## STSAFE-A Pairing Keys

The Pairing Keys shall be implemented according to the MCU capabilities.

The SE\_STSAFEA\_API only offer the capability to send the serial number to the crypto interface implementation to load the device Pairing Keys.

The stsafea\_crypto\_interface.c propose an example of the cryptography implementation and use only default keys.

The StsafeA\_Init will call the StSafeA\_HostKeys\_Init() but does not provide the STSAFE-A serial number. This could be enough if they is not derivation of the Pairing Keys using the STSAFE-A serial Number.

If the STSAFE-A Serial Number is necessary, the SE\_STSAFEA\_API is calling the StSafeA\_Crypto\_GetKeys() with the STSAFE-A Serial Number before using the secure channel.

This will give a place holder for the user implementation to use STSAFE-A Serial Number to get the Pairing Keys.

The Pairing keys handling is still the responsibility of the user integration.

## STSAFE-A Auto management

The SE\_STSAFEA\_API Auto Management mechanism is helper to use single function to use STSAFE-A private Key slots for authentication.

For that, the SE\_STSAFEA\_API mandate a read/write memory region with 2 bytes available.

This could be added within custom memory mapping.

To configure the Auto Management, you can use either #define to fix the default value for auto management system.

STSAFEA\_AUTO\_MANAGED\_REGION

STSAFEA\_AUTO\_MANAGED\_CERTIFICATE\_OFFSET

STSAFEA\_AUTO\_MANAGED\_KEY\_OFFSET

Or to use the SE\_SetAutoManagement() function to configure the auto management.

The Auto Management apply only to:

* SE\_GetActiveCertificate
* SE\_SetActiveCertificate
* SE\_GetActiveKey
* SE\_SetActiveKey

It is then possible to automatically use the correct key or certificate with single function relying on the SE\_GetActivexxx function.

## I2C configuration

The SE\_STSAFEA\_API request I2C address at Init and the default I2C address for example code is managed by #define

STSAFEA\_DEFAULT\_I2C\_ADDR

## CMake configuration

SE\_STSAFEA\_API comes with CMake build system for Linux example.

For CMake usage, the previous define can be defined with the following cmake command line argument:

STSAFE\_I2C\_ADDR

STSAFEA\_AUTO\_REGION

STSAFEA\_AUTO\_CERT

STSAFEA\_AUTO\_KEY

## Linux specific behavior

In Linux system, some traces can be activated by creating the file /var/log/stsafe.log with the correct access condition (depends on which user is starting the application using STSAFE-A).

# Crypto Library

For secure channel between host and STSAFE-A110, AES-CBC and AES-CMAC is mandatory.

The X-CUBE-SAFEA1 expose dedicated API for this cryptography.

Therefore, a crypto stack is mandatory.

The SE\_STSAFEA\_API example comes with build system for MbedTLS.

OpenSSL support to come.

## MbedTLS

Integration of MbedTLS is using CMake MbedTLS build system.

The SE\_STSAFEA\_API package provides a MbedTLS configuration file and expose 2 CMake variable to activate it.

* USE\_MBEDTLS=1
* MBEDTLS\_BASEDIR=<mbedtls source dir>

The MBEDTLS\_BASEDIR shall point on a SE\_STSAFEA\_API subdirectory (you can make a symbolic link in the SE\_STSAFEA\_API directory)

In the build directory,

Cmake .. -DUSE\_MBEDTLS=1 -DMBEDTLS\_BASEDIR=<path to mbedtls>

For instance, <path to mbedtls> = $PWD/../mbedtls

MBEDTLS is also mandatory to build the application tools mainly for certificates handling.

## OpenSSL

OpenSSL support will use version 1.1.1 API.

# API description

## Flags and Structures descriptions

### Flags

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Value** | **Function** | **Remarks** |
| SE\_INIT\_TEST\_ECHO | 1 | SE\_Init | Activate echo testing at init |
| SE\_INIT\_TEST\_PAIRING | 2 | SE\_Init | Activate pairing key testing at init – increase the pairing sequence counter |
| SE\_ACCESS\_RIGHT\_LOCK | 0x10 | SE\_ReadRight  SE\_UpdateRight | Lock region permission update |
| SE\_ACCESS\_RIGHT\_ALWAYS | 0 | SE\_ReadRight  SE\_UpdateRight | Set region permission to always |
| SE\_ACCESS\_RIGHT\_HOST | 1 | SE\_ReadRight  SE\_UpdateRight | Set region permission to pairing CMAC |
| SE\_ACCESS\_RIGHT\_NEVER | 7 | SE\_ReadRight  SE\_UpdateRight | Set region permission to never (permission removed) |

### Error Codes

|  |  |  |
| --- | --- | --- |
| **Name** | **Value** | **Description** |
| SE\_OK | 0 | No error |
| SE\_COMMAND\_ERROR | -1 | STSAFE-A reports an error while executing command |
| SE\_INVALID\_STATE\_ERROR | -2 | SE\_STSAFEA\_API not started – call SE\_Init first |
| SE\_INVALID\_PARAMS\_ERROR | -3 | Invalid paremeter passed to the function |
| SE\_INVALID\_SIG\_ERROR | -4 | Signature verificiation failed to invalid signature |
| SE\_BUFFER\_TOO\_SHORT\_ERROR | -5 | Structure passed to short, expected value returned |

### Structures

### SE\_Infot\_t

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| productData | SE\_Product\_Data\_t | Product data extracted from STSAFE-A |
| i2cParams | SE\_I2C\_Params\_t | I2C configuration |
| pairing | SE\_Pairing\_Info\_t | Pairing key status |
| envelop | SE\_Envelop\_Info\_t | Envelop key status |
| lifecycle | uint8\_t | STSAFE-A lifecycle status. Check SE\_State\_t |

### SE\_Product\_Data\_t

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| InOutBufferSize | uint16\_t | STSAFE-A I2C buffer |
| NVMSize | uint16\_t | STSAFE-A memory size |
| AtomicityBuffer | uint16\_t | STSAFE-A atomicity buffer for memory update |
| serial | uint8\_t | Serial Number offset in the product data raw\_data |
| raw\_data | uint8\_t[100] | Raw\_data output from StSafeA\_ProductDataQuery() |
| raw\_size | uint8\_t | Size of the product data in the raw\_data |

### SE\_I2C\_Params\_t

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| I2cAddress | uint8\_t | I2C address of STSAFE-A on 7bits |
| LowPowerModeConfig | uint8\_t | STSAFE-A low power mode |
| LockConfig | uint8\_t | Lock of the I2C params |

### SE\_Pairing\_Info\_t

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| present | uint8\_t | 1 if STSAFE-A pairing is done |
| counter | uint32\_t | If STSAFE-A pairing present is 1 value of the pairing sequence counter |

### SE\_Envelop\_Info\_t

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| presence0 | uint8\_t | 1 is Envelop Key slot 0 is present |
| size0 | uint8\_t | Size of Envelop Key slot 0 is present – See SE\_Wrap\_Type\_t for details |
| presence1 | uint8\_t | 1 is Envelop Key slot 1 is present |
| size1 | uint8\_t | Size of Envelop Key slot 1 is present – See SE\_Wrap\_Type\_t for details |

### SE\_State\_t

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| SE\_STATE\_OPERATIONNAL | Enum | STSAFE-A is operational |
| SE\_STATE\_LOCKED | Enum | STSAFE-A is locked and shall be unlocked before usafe |
| SE\_STATE\_TERMINATED | Enum | STSAFE-A is terminated |
| SE\_STATE\_UNKNOWN | Enum | Error while reading STSAFE-A State |

### SE\_Cmd\_t

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| id | uint8\_t | Command ID |
| auth | uint8\_t | Command authentication configuration |
| enc | uint8\_t | Command encryption policy |

### SE\_Region\_t

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| index | uint8\_t | Region index |
| size | uint16\_t | Region size |
| read\_ac | uint8\_t | Read permission |
| read\_update | uint8\_t | Read permission update flag |
| write\_ac | uint8\_t | Write permission |
| write\_update | uint8\_t | Write permission update flag |
| OneWay | uint32\_t | Value of the one-way counter if region has one-way counter |
| type | uint8\_t | 1 if region has an one-way counter |

### SE\_SerialFormat\_t

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| SE\_SERIAL\_FORMAT\_RAW | Enum | Output of SE\_GetSerial is 9 bytes RAW Serial Number |
| SE\_SERIAL\_FORMAT\_HEX | Enum | Output of SE\_GetSerial is 19 char string including null char terminated of hexadecimal representation of the Serial Number |

### SE\_Slot\_t

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| SE\_SLOT\_PREPROVISIONNED | Enum | Use Private Key Slot 0 – preprovisionned key |
| SE\_SLOT\_PERMANENT | Enum | Use Private Key Slot 1 – permanent key storage |
| SE\_SLOT\_EPHEMERAL | Enum | Use Private Key Slot FF – RAM key storage (not permanent) |

### SE\_Hash\_Type\_t

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| SE\_HASH\_SHA\_256 | Enum | Not use |
| SE\_HASH\_SHA\_384 | Enum | Not use |

### SE\_Key\_Type\_t

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| SE\_KEY\_TYPE\_NIST\_256 | Enum | Key type for key creation |
| SE\_KEY\_TYPE\_NIST\_384 | Enum | Key type for key creation |
| SE\_KEY\_TYPE\_BP\_256 | Enum | Key type for key creation |
| SE\_KEY\_TYPE\_BP\_384 | Enum | Key type for key creation |

### SE\_Security\_Level\_t

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| SE\_SECURITY\_LEVEL\_NONE | Enum | No secure channel for commands (override by commands configuration or region permission) |
| SE\_SECURITY\_LEVEL\_C\_PAIRING | Enum | All commands will have a Command MAC |
| SE\_SECURITY\_LEVEL\_R\_PAIRING | Enum | All commands will have a Command MAC and Response MAC |

### SE\_Wrap\_Type\_t

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| SE\_WRAP\_KEY\_128 | Enum | Envelop key size AES 128 bits |
| SE\_WRAP\_KEY\_256 | Enum | Envelop key size AES 256 bits |

## Init and status commands

### SE\_Init

|  |  |
| --- | --- |
| **Prototype** | int32\_t SE\_Init(uint32\_t flag, SE\_Security\_Level\_t sec\_level, uint32\_t i2c\_addr) |
| **Parameter** | flag: SE\_INIT\_TEST\_ECHO and/or SE\_INIT\_TEST\_PAIRING  sec\_level: initial security level  i2c\_addr: STSAFEA\_A I2C address on 7bits (default STSAFE-A I2C addr is 0x20) |
| **Return Value** | SE\_OK if no error, SE\_COMMAND\_ERROR otherwise |
| **Description** | Start the STSAFE API  If flag has SE\_INIT\_TEST\_ECHO: perform a basic echo command.  If flag has SE\_INIT\_TEST\_PAIRING: check if pairing keys are defined, if yes, execute wrap/unwrap commands to validated both pairing keys.  Get the private key slot key size.  Get the auto management status from default value.  Configure the security level for all following calls to the API |

### SE\_SetSecurity

|  |  |
| --- | --- |
| **Prototype** | int32\_t SE\_SetSecurity(SE\_Security\_Level\_t level) |
| **Parameter** | Level: security level |
| **Return Value** | SE\_OK |
| **Description** | Configure the security level for all following calls to the API |

### SE\_GetSerial

|  |  |
| --- | --- |
| **Prototype** | int32\_t SE\_GetSerial(uint8\_t \*serial, SE\_SerialFormat\_t format) |
| **Parameter** | Serial: char table to get the serial number shall be 9 or 19 bytes  Format: select SE\_SERIAL\_FORMAT\_RAW or SE\_SERIAL\_FORMAT\_HEX |
| **Return Value** | SE\_OK if no error, SE\_COMMAND\_ERROR otherwise |
| **Description** | Get the STSAFE-A Unique Identifier on 9 bytes.  If SE\_SERIAL\_FORMAT\_RAW, serial parameter shall be 9 bytes.  If SE\_SERIAL\_FORMAT\_HEX, serial parameter shall be 19 bytes and the hexadecimal representation the 9 bytes |

### SE\_GetKeySize

|  |  |
| --- | --- |
| **Prototype** | int32\_t SE\_GetKeySize(uint8\_t slot) |
| **Parameter** | Slot: select the slot to query |
| **Return Value** | Key slot size in bits or 0 if key is not present |
| **Description** | Get the private key slot size if the key is present or 0 otherwise |

### SE\_GetCommandStatus

|  |  |
| --- | --- |
| **Prototype** | int32\_t SE\_GetCommandStatus(SE\_Cmd\_t \*cmd, uint32\_t \*size) |
| **Parameter** | Cmd: SE\_Cmd\_t structures  Size : pointer on size fo the cmd size |
| **Return Value** | SE\_OK if no error, SE\_COMMAND\_ERROR otherwise |
| **Description** | Size represents the size of the cmd table.  If size is 0 and cmd is NULL, the function returns the current cmd table size.  This could help to allocate the cmd table to the correct size.  If size is not 0 and cmd is not NULL  Fill up the table of SE\_Cmd\_t structures with commands authorization and encryption configuration (see stsafea\_desribe application for more information) |

### SE\_GetMapping

|  |  |
| --- | --- |
| **Prototype** | int32\_t SE\_GetMapping(SE\_Region\_t \*regions, uint32\_t \*nb\_region) |
| **Parameter** | Regions: table of nb\_region SE\_Region\_t  Nb\_region: number of regions in the regions table. |
| **Return Value** | SE\_OK if no error, SE\_COMMAND\_ERROR otherwise |
| **Description** | If nb\_region is 0, the number of the memory region is return in nb\_region to allocate the regions table to the correct size.  Get the STSAFE-A memory mapping with per region the size, the attributes, and the access permissions |

### SE\_GetRandom

|  |  |
| --- | --- |
| **Prototype** | int32\_t SE\_GetRandom(void \*ctx, uint8\_t \*data, size\_t size) |
| **Parameter** | Ctx: Unused  Data: table to store the random value  Size: requested size |
| **Return Value** | SE\_OK if no error, -1 otherwise |
| **Description** | Fill up the data table with random value generated by STSAFE-A |

### SE\_GetInfo

|  |  |
| --- | --- |
| **Prototype** | int32\_t SE\_GetInfo(SE\_Info\_t \*se\_info) |
| **Parameter** | se\_info: get STSAFE-A configuration info |
| **Return Value** | SE\_OK if no error, -1 otherwise |
| **Description** | Fill up the se\_info structure including  Product Data with Serial Number  I2CParameters status  Pairing Key slot status  Envelop Key status  LifeCycle status |

## Management command

### SE\_Reset

|  |  |
| --- | --- |
| **Prototype** | int32\_t SE\_Reset(uint32\_t type) |
| **Parameter** | Type: type of reset |
| **Return Value** | SE\_OK if no error, -1 otherwise |
| **Description** | If type is 0 execute the STSAFE-A Reset command  If type is 1 execute an HW reset (not implemented) |

### SE\_Hibernate

|  |  |
| --- | --- |
| **Prototype** | int32\_t SE\_Hibernate(uint32\_t type) |
| **Parameter** | Type: type of hibernation |
| **Return Value** | SE\_OK if no error, -1 otherwise |
| **Description** | If type is 1 hibernate with wakeup event from reset only  Otherwise  Hibernate with wakeup event from I2C start condition or reset |

### SE\_Echo

|  |  |
| --- | --- |
| **Prototype** | int32\_t SE\_Echo(uint8\_t \*input, size\_t size, uint8\_t \*output) |
| **Parameter** | Input: buffer to be echoed  Size: size of data to echo  Output: return of the echo command |
| **Return Value** | SE\_OK if no error, -1 otherwise |
| **Description** | Basic test function to validate STSAFE-A communication |

### SE\_Refresh

|  |  |
| --- | --- |
| **Prototype** | int32\_t SE\_Refresh(void) |
| **Parameter** | None |
| **Return Value** | SE\_OK if no error, -1 otherwise |
| **Description** | Refresh internal structure initialized by SE\_Init ( this will refresh the internal cached data after an unlock of the STSAFE-A for instance) |

### SE\_Raw

|  |  |
| --- | --- |
| **Prototype** | int32\_t SE\_Raw(uint8\_t \*cmd, size\_t cmd\_size, uint8\_t \*rsp, size\_t \*rsp\_size) |
| **Parameter** | Cmd: command buffer (without CRC)  Cmd\_size: size of command buffer  Rsp: response buffer  Rsp\_size: response buffer size |
| **Return Value** | SE\_OK if no error, -1 otherwise |
| **Description** | Allow to run RAW commands to STSAFE-A |

## Enablement commands

### SE\_GetState

|  |  |
| --- | --- |
| **Prototype** | int32\_t SE\_GetState(SE\_State\_t \*state) |
| **Parameter** | State: STSAFE-A state |
| **Return Value** | SE\_OK if no error, SE\_COMMAND\_ERROR otherwise |
| **Description** | Get the STSAFE-A status and set it in the state parameter  State can be:   * SE\_STATE\_OPERATIONNAL: STSAFE-A is functional * SE\_STATE\_TERMINATED: STSAFE-A is in terminated state and no longer usable * SE\_STATE\_LOCKED: STSAFE-A is locked, and need unlock * SE\_STATE\_UNKNOWN: Invalid state |

### SE\_Unlock

|  |  |
| --- | --- |
| **Prototype** | int32\_t SE\_Unlock(uint8\_t \*password) |
| **Parameter** | Password: 16 bytes table with STSAFE-A password |
| **Return Value** | SE\_OK if no error, SE\_COMMAND\_ERROR or SE\_INVALID\_PARAMS\_ERROR |
| **Description** | Use the password to unlock the STSAFE-A if STSAFE-A state is SE\_STATE\_LOCKED.  If password is invalid SE\_INVALID\_PARAMS\_ERROR is return |

### SE\_Pairing

|  |  |
| --- | --- |
| **Prototype** | int32\_t SE\_Pairing(uint8\_t \*cipher, uint8\_t \*mac) |
| **Parameter** | Cipher: 16 bytes Cipher Key  Mac: 16 bytes Mac Key |
| **Return Value** | SE\_OK |
| **Description** | Check if pairing keys is present, if not, set the pairing keys to STSAFE-A |

### SE\_GetPairingStatus

|  |  |
| --- | --- |
| **Prototype** | int32\_t SE\_GetPairingStatus(uint32\_t \*presence, uint32\_t \*counter) |
| **Parameter** | Presence: presence flag for pairing keys  Counter: pairing keys sequence counter |
| **Return Value** | SE\_OK if no error, SE\_COMMAND\_ERROR otherwise |
| **Description** | Get the Pairing keys presence and if yes, get the sequence counter for the secure channel |

### SE\_SetPairingKeys

|  |  |
| --- | --- |
| **Prototype** | int32\_t SE\_SetPairingKeys(uint8\_t \*keys) |
| **Parameter** | Keys: 32 bytes of MAC key and Cipher key for pairing |
| **Return Value** | SE\_OK if no error, SE\_COMMAND\_ERROR otherwise |
| **Description** | Set the Pairing key in STSAFE-A. |

## Memory partition access

### SE\_Read

|  |  |
| --- | --- |
| **Prototype** | int32\_t SE\_Read(uint8\_t region, uint16\_t offset, uint16\_t size, uint8\_t \*buf) |
| **Parameter** | Region: region to read  Offset: offset in the region to start reading  Size: size of data to read  Buf: data buffer of size <size> |
| **Return Value** | SE\_OK if no error, SE\_COMMAND\_ERROR otherwise |
| **Description** | Read the memory region. This API will split the read if needed. Buf shall be of expected size.  If offset is out of region or size is too big for the region, SE\_COMMAND\_ERROR is returned. |

### SE\_ReadRight

|  |  |
| --- | --- |
| **Prototype** | int32\_t SE\_ReadRight(uint8\_t region, uint16\_t offset, uint32\_t update, uint16\_t size, uint8\_t \*buf) |
| **Parameter** | Region: region to read  Offset: offset in the region to start reading  Update: New access right to set on read permission  Size: size of data to read  Buf: data buffer of size <size> |
| **Return Value** | SE\_OK if no error, SE\_COMMAND\_ERROR otherwise |
| **Description** | Read the memory region. This API will split the read if needed. Buf shall be of expected size.  If offset is out of region or size is too big for the region, SE\_COMMAND\_ERROR is returned. |

### SE\_Update

|  |  |
| --- | --- |
| **Prototype** | int32\_t SE\_Update(uint8\_t region, uint16\_t offset, uint16\_t size, uint8\_t \*data) |
| **Parameter** | Region: region to write  Offset: offset in the region to start writing  Size: size of data to write  Buf: data buffer |
| **Return Value** | SE\_OK if no error, SE\_COMMAND\_ERROR otherwise |
| **Description** | Write to the memory region. This API will split the writing if needed.  If offset is out of region or size is too big for the region, SE\_COMMAND\_ERROR is returned. |

### SE\_UpdateRight

|  |  |
| --- | --- |
| **Prototype** | int32\_t SE\_UpdateUpdate(uint8\_t region, uint16\_t offset, uint32\_t update, uint16\_t size, uint8\_t \*data) |
| **Parameter** | Region: region to write  Offset: offset in the region to start writing  Update: new access right to set on update permission  Size: size of data to write  Buf: data buffer |
| **Return Value** | SE\_OK if no error, SE\_COMMAND\_ERROR otherwise |
| **Description** | Write to the memory region. This API will split the writing if needed.  If offset is out of region or size is too big for the region, SE\_COMMAND\_ERROR is returned. |

## Local Secure Storage commands

### Se\_GenerateWrapKey

|  |  |
| --- | --- |
| **Prototype** | int32\_t SE\_GenerateWrapKey(int8\_t slot, SE\_Wrap\_Type\_t type) |
| **Parameter** | Slot: local envelope key slot to use (SE\_SLOT\_0 or SE\_SLOT\_1)  Type: size of the key to generate |
| **Return Value** | SE\_OK if no error, SE\_COMMAND\_ERROR otherwise |
| **Description** | Generate an envelope key on selected slot of selected size.  SE\_WRAP\_KEY\_256 or SE\_WRAP\_KEY\_128 |

### SE\_Wrap

|  |  |
| --- | --- |
| **Prototype** | int32\_t SE\_Wrap(uint8\_t slot, uint8\_t \*data, uint16\_t size, uint8\_t \*enc\_data) |
| **Parameter** | Slot: local envelope key slot to use (SE\_SLOT\_0 or SE\_SLOT\_1)  Data: data to wrap, shall be multiple of 8 bytes  Size: size of data to wrap  Enc\_data: table for encrypted data, the size of enc\_data shall be size + 8 bytes |
| **Return Value** | SE\_OK if no error, SE\_COMMAND\_ERROR otherwise |
| **Description** | Perform an AES Key wrap of the data. The data shall be multiple of 8 bytes and the enc\_data shall be the size of data + 8 bytes.  It is recommended to wrap only small amount of data like AES 256 key for instance  This API mandates the pairing keys to be configured |

### SE\_Unwrap

|  |  |
| --- | --- |
| **Prototype** | int32\_t SE\_Unwrap(uint8\_t slot, uint8\_t \*enc\_data, uint16\_t size, uint8\_t \*data) |
| **Parameter** | Slot: local envelope key slot to use (SE\_SLOT\_0 or SE\_SLOT\_1)  Enc\_data: envelope generated by SE\_Wrap  Size: size of enc\_data  Data: table for clear data, the size of data shall be size - 8 bytes |
| **Return Value** | SE\_OK if no error, SE\_COMMAND\_ERROR otherwise |
| **Description** | Perform an AES Key unwrap of the envelope data.  This API mandates the pairing keys to be configured. |

## Asymmetric primitive

### SE\_VerifySignature

|  |  |
| --- | --- |
| **Prototype** | int32\_t SE\_VerifySignature(SE\_Key\_Type\_t k\_type, uint8\_t \*hash, size\_t hash\_size, uint8\_t \*pub\_X, uint8\_t \*pub\_Y, uint8\_t \*R, uint8\_t \*S) |
| **Parameter** | K\_type: type of the public key for signature verification  Hash: hash for signature  Hash\_size: size of the hash  X: Public key X parameter, size shall be 32 if key is 256 bits or 48 if key is 384 bits  Y: Public key Y parameter, size shall be 32 if key is 256 bits or 48 if key is 384 bits  R: Signature R parameter, size shall be 32 if key is 256 bits or 48 if key is 384 bits  S: Signature S parameter, size shall be 32 if key is 256 bits or 48 if key is 384 bits |
| **Return Value** | SE\_OK if no error, SE\_INVALID\_SIG\_ERROR is signature is not correct or SE\_COMMAND\_ERROR is command fail |
| **Description** | Verify the ECC signature of the Hash according to the Public Key and Signature provided. |

### SE\_GenerateSignature

|  |  |
| --- | --- |
| **Prototype** | int32\_t SE\_GenerateSignature(uint8\_t slot, const uint8\_t \*hash, uint16\_t hash\_size, uint8\_t \*R, uint8\_t \*S) |
| **Parameter** | Slot: Private key slot to use (SE\_SLOT\_0 or SE\_SLOT\_1)  Hash: hash to sign  Hash\_size: size of hash to sign  R,S: data array for signature, shall be 32 bytes each if signing key size is 256 bits and 48 bytes each if signing key size is 384 |
| **Return Value** | SE\_OK if no error, SE\_COMMAND\_ERROR otherwise |
| **Description** | Request STSAFE-A to generate a signature on the provided hash.  If hash size is not the same than private key size, the hash will be padded or truncated. |

### SE\_GenerateKeyPair

|  |  |
| --- | --- |
| **Prototype** | int32\_t SE\_GenerateKeyPair(SE\_Slot\_t slot, SE\_Key\_Type\_t type, uint8\_t \*pub\_x, uint8\_t \*pub\_y) |
| **Parameter** | Slot: Slot to use to generate the key (SE\_SLOT\_PERMANENT or SE\_SLOT\_EPHEMERAL)  Type: key curve to use  X,Y: X, Y coordinate of the public key. |
| **Return Value** | SE\_OK if no error, SE\_COMMAND\_ERROR or SE\_INVALID\_PARAMS\_ERROR |
| **Description** | Generate a new Key Pair on selected slot.  If Key Pair shall be used for authentication, the slot shall be SE\_SLOT\_PERMANENT, if Key Pair shall be used for ECDH, the slot shall be SE\_SLOT\_EPHERMERAL.  If the type is not a supported type (SE\_KEY\_TYPE\_NIST\_256, SE\_KEY\_TYPE\_NIST\_384, SE\_KEY\_TYPE\_BP\_256, SE\_KEY\_TYPE\_BP\_384) SE\_INVALID\_PARAMS\_ERROR is return.  X and Y table will receive the public part of the generated key.  X and Y shall be allocated array of 32 bytes if key is 256 bits and 48 bytes if key is 384 bits |

### SE\_DeriveSharedSecret

|  |  |
| --- | --- |
| **Prototype** | int32\_t SE\_DeriveSharedSecret(SE\_Slot\_t slot, uint8\_t \*x, uint8\_t \*y, uint8\_t \*z, uint16\_t size) |
| **Parameter** | Slot: slot to use for the key derivation (SE\_SLOT\_EPHEMERAL or SE\_SLOT\_PERMANENT)  x, y: public key for ECDH  z: shared secret  Size: size of X, Y parameter |
| **Return Value** | SE\_OK if no error, SE\_COMMAND\_ERROR otherwise |
| **Description** | Generate shared secret with internal private key and provided public key.  Size of x, y parameters shall be provided.  ECDH can be done only on EPHEMERAL slot dedicated for ECDHE function or on PERMANENT slot which is mapped to STSAFE-A slot 1 |

## Application level Key and certificate management

### SE\_StoreCertificate

|  |  |
| --- | --- |
| **Prototype** | int32\_t SE\_StoreCertificate(uint8\_t id, uint8\_t \*buf, uint16\_t size) |
| **Parameter** | Id: region to store the certificate  Buf: data buffer with certificate in DER  Size: size of the certificate to store |
| **Return Value** | SE\_OK if no error, SE\_COMMAND\_ERROR otherwise |
| **Description** | Store a certificate in a memory region |

### SE\_GetCertificate

|  |  |
| --- | --- |
| **Prototype** | int32\_t SE\_GetCertificate(uint8\_t region, uint8\_t \*buf, size\_t \*size) |
| **Parameter** | Region: region to read a certificate  Buf: buffer to store the extracted certificate  Size: initial size of the Buf |
| **Return Value** | SE\_OK if no error, SE\_COMMAND\_ERROR otherwise |
| **Description** | Try to read a certificate from STSAFE-A region.  The size of the certificate is returned in the size paramaters |

### SE\_SetActiveKey

|  |  |
| --- | --- |
| **Prototype** | int32\_t SE\_SetActiveKey(uint8\_t id) |
| **Parameter** | Id: 0 or 1 |
| **Return Value** | SE\_OK if no error, SE\_COMMAND\_ERROR otherwise |
| **Description** | Set the active private key in the auto managed system.  Represent the STSAFE-A private key slot and shall be 0 or 1 |

### SE\_GetActiveKey

|  |  |
| --- | --- |
| **Prototype** | int32\_t SE\_GetActiveKey(uint8\_t \*id) |
| **Parameter** | Id: active key |
| **Return Value** | SE\_OK if no error, SE\_COMMAND\_ERROR otherwise |
| **Description** | Get the current active key in the auto managed system. |

### SE\_GetActiveCertificate

|  |  |
| --- | --- |
| **Prototype** | int32\_t SE\_GetActiveCertificate(uint8\_t \*buf, size\_t \*size) |
| **Parameter** | Buf: buffer to store the certificate  Size: initial size of the buffer |
| **Return Value** | SE\_OK if no error, SE\_COMMAND\_ERROR otherwise |
| **Description** | Get the active certificate based on the auto management system. |

### SE\_SetActiveCertificate

|  |  |
| --- | --- |
| **Prototype** | int32\_t SE\_SetActiveCertificate(uint8\_t id) |
| **Parameter** | Id: STSAFE-A memory region containing the new active certificate |
| **Return Value** | SE\_OK if no error, SE\_COMMAND\_ERROR otherwise |
| **Description** | Set in the auto managed system the new active certificate. |

# Examples Description

Examples application provide a set of applicative features.

Each application has a common set of option:

-a, --auto : auto management region definition

-i, --addr : STSAFE-A I2C addr

## stsafea\_describe

stsafea\_describe application provide general information on STSAFE-A:

* Serial Number
* Pairing key status
* STSAFE-A State
* Slot 0 Key Size
* Slot 1 Key Size
* Memory Mapping
* Commands authorization
* Region 0 Certificate info
* Auto managed Certificate info

Extra option:

-p, --pairing: activate pairing testing at init.

-l, --loop: execute in loop commands using secure channel

-v, --verbose: activate SE\_STSAFEA\_API trace if SE\_STSAFEA\_API is compiled with Debug option

Example of result of stsafea\_describe on STSAFA110DFSPL02 sample:

