AN12413

SE050 APDU Specification

Rev. 2.12 — 24 March 2021 Application note

Document information

Information	Content
Keywords	SE050 Applet, Internet of Things, Secure Element
Abstract	This document provides the API description of the EdgeLock™ SE050 Plug & Trust secure element family.

Revision history

Revision history

Rev	Date	Description
2.12	2021-03-24	Integrated errata into the Revision History table and removed Errata chapter.
2.11	2020-07-16	Minor updates
2.10	2020-06-22	Updated Section 4.7.2
2.9	2020-03-27	Renamed FastSCP to ECKey session and minor updates.
2.8	2019-12-19	Minor updates
2.7	2019-12-17	Minor updates
2.6	2019-12-12	Minor updates
2.5	2019-12-03	Added information for FIPS compliance. • Applets prior to version 3.4.0 allow to set userIDs from 1 byte to 16 bytes. • Applets prior to version 3.4.0 allow to apply policies to Authentication Objects as applicable for regular Secure Objects (see Object policies and Authentication object policies). • Applets prior to version 3.4.0 allow to use EC key pairs for both ECDSA and ECDH; from 3.4.0 onwards, EC key pairs can only be used for ECDSA when CONFIG_FIPS_MODE_DISABLED is set to 0.
2.4	2019-11-27	Added errata on HKDF to applet 3.1.0 specification. Removed Expand-only mode from HKDF (see [Errata]).
2.3	2019-07-01	Prepared for release of applet 3.1.0

1 Introduction

1.1 Context

SE050 is designed to be used as a part of an IoT system. It works as an auxiliary security device attached to a host controller. The host controller communicates with SE050 through an I²C interface (with the host controller being the master and the SE050 being the slave). Besides the mandatory connection to the host controller, the SE050 device can optionally be connected to a sensor node or similar element through a separate I²C interface. In this case, the SE050 device is the master and the sensor node the slave. Lastly, SE050 has a connection for a native contactless antenna, providing a wireless interface to an external device such as a smartphone.

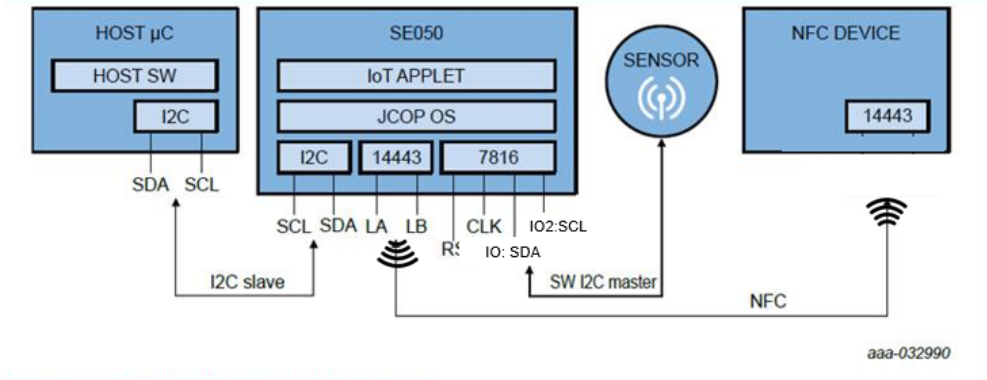

Figure 1. SE050 solution block diagram

The SE050 provides a wide range of (cryptographic) possibilities. Note that users need to be aware of the cryptographic principles when using the functionality of SE050 for the intended use cases.

2 SE050 card architecture

2.1 Security Domain layout

NXP is in control of the Secondary Security Domain (SSD) which holds the SE050 IoT applet. In addition, mandated DAP is set in NXP SSD, so only signed applets can be loaded onto the device. The AID of the NXP IoT SSD is D276000085304A434F9003.

2.2 Operating system

The operating system used is JCOP 4.7.

2.3 Applet

The instance AID for SE050 IoT applet - pre-provisioned by NXP - is A000000396545300000001030000000000.

The supported applet versions are 3.1.0, 3.1.1 and 3.6.0.

The APDU buffer size is 894 bytes.

3 SE050 IoT applet functionality overview

This section provides an overview of the functionalities of the SE050 IoT applet.

3.1 Supported functionality

The SE050 IoT applet supports:

- Generic module management
  - Lifecycle management
  - Session management
  - Timer functionality
  - Access control
  - Secure import/export of keys or files
- Applet Secure Channel management
  - AESKey sessions (previously called SCP03)
  - ECKey sessions (previously called FastSCP)
- Random number generation
- Key management (ECC, RSA, AES, DES, etc.): write, read, lock, delete
- Elliptic curve cryptographic operations
- RSA cryptographic operations
- AES/DES cryptographic operations (AES ECB, CBC, CTR)
- Binary file creation and management
- UserID creation and management
- Monotonic counter creation and management
- PCR creation and management
- Hash operations
- Message authentication code generation
  - CMAC
  - HMAC
- Key derivation functionality
  - HKDF
  - PBKDF2 using HMAC SHA1
- Specific use case support
  - TLS PSK master secret calculation
  - MIFARE DESFire protocol support
  - I2C Master support

3.2 SE050 Secure Objects

3.2.1 Classes

The SE050 has one base object type called *Secure Object*. A Secure Object can be derived to classes depicted in Figure 2:

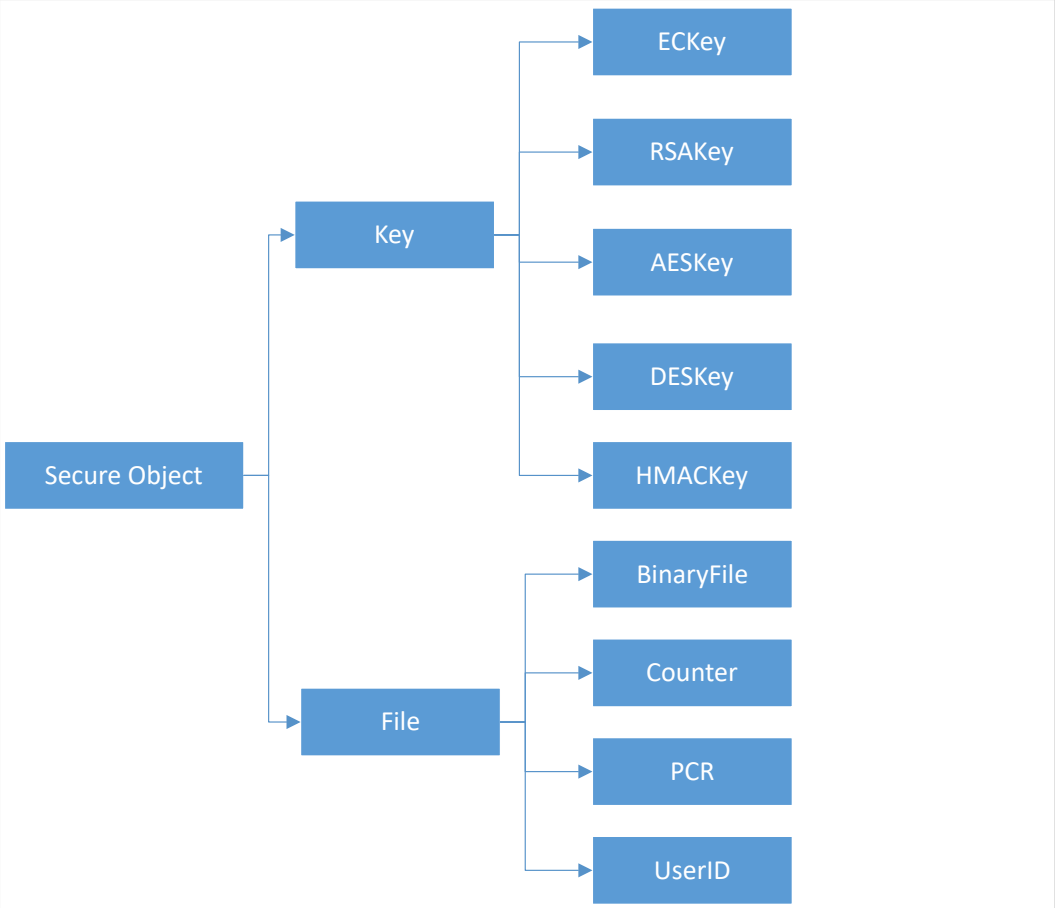

Figure 2. SE050 Secure Object structure

3.2.1.1 ECKey

An ECKey object is any elliptic curve key type (key pair/private key/public key), either transient (Cleared on Deselect (CoD)) or persistent. ECKey objects are linked to one of the supported EC curves (listed in Table 37).

EC private keys are always represented in a byte array that is exactly equal to EC curve bit size (e.g., 32 bytes for private keys on NIST P-256).

EC public keys are represented in uncompressed form for all curves in Weierstrass form; i.e., a byte array starting with 0x04 followed by the X and Y coordinates concatenated. Both X and Y are again exactly equal to the EC curve bit size.

For the Edwards curve 25519 (ECC_ED_25519), public and private keys are exactly 32 bytes long. This curve needs to be used for signature operations (sign/verify).

For the Montgomery curve 25519 (ECC_MONT_DH_25519), public and private keys are exactly 32 bytes long. This curve needs to be used for DH operations.

When the rules for the length of the keys are not strictly applied, using the stored key can lead to a system reset of the device.

Table 1. Supported EC curves

Name	Weierstrass	Private key byte length	Public key byte length	Shared secret length	Remarks
UNUSED	-	-	-	-	
NIST_P192	Y	24	49	24	
NIST_P224	Y	28	57	28	
NIST_P256	Y	32	65	32	
NIST_P384	Y	48	97	48	
NIST_P521	Y	66	133	66	
Brainpool160	Y	20	41	20	
Brainpool192	Y	24	49	24	
Brainpool224	Y	28	57	28	
Brainpool256	Y	32	65	32	
Brainpool320	Y	40	81	40	
Brainpool384	Y	48	97	48	
Brainpool512	Y	64	129	64	
Secp160k1	Y	20	41	20	
Secp192k1	Y	24	49	24	
Secp224k1	Y	28	57	28	
Secp256k1	Y	32	65	32	
TPM_ECC_BN_P256	Y	32	65	32	Fp256BN Barreto-Naehrig curve.
ID_ECC_ED_25519	N	32	32	32	Edwards curve 25519 to be used for EdDSA sign/verify operations. See Section 7 for correct byte order..
ID_ECC_MONT_DH_25519	N	32	32	32	Montgomery curve 25519 to be used for shared secret generation. See Section 7 for correct byte order.

3.2.1.2 RSAKey

An RSAKey object is any RSA key type (key pair/private key/public key), either transient (Cleared on Deselect) or persistent. The private key can be in CRT or in raw format.

In CRT format, the key components must match the size of the key type, e.g. for RSA2048, each component must be 2048 bit (256 bytes). In raw format, keys must match the key type.

3.2.1.3 AESKey

An AESKey object is any AES key of size 128, 192 or 256 bit, either transient (Cleared on Deselect) or persistent.

3.2.1.4 DESKey

A DESKey object is any DES key, either transient (Cleared on Deselect) or persistent.

DESKey objects store the keys including parity bits, so the length is either 8, 16 or 24 bytes respectively for DES, 2-key 3DES and 3-key 3DES. The value of the parity bits is not checked inside the SE050.

3.2.1.5 HMACKey

An HMACKey object is a secret of any length, 1 up to 256 bytes. Typically, it is used as input for message authentication codes or key derivation functions when the key material is not 16 or 32 bytes in length. It can be either transient or persistent.

3.2.1.6 BinaryFile

A BinaryFile object is a file containing a byte array of a specific length (minimum 1 byte). Files are initialized by default with all 0x00. It can be either transient (Cleared on Deselect) or persistent.

The transient binary files are reset to zero on deselection or actual reset. The maximum file size is 0x7FFF bytes.

3.2.1.7 Counter

A counter object is a monotonic counter, either transient (Cleared on Deselect) or persistent. A monotonic counter can only be incremented and not be decremented to a lower value. Note that transient counters are an exception as the value is reset to all zeroes on a deselect. Its length is 1 up to 8 bytes.

3.2.1.8 PCR

A Platform Configuration Register (PCR) object is a 32-byte array that holds the value of a SHA256. PCRs can be either persistent or transient. Transient PCRs are reset on deselect of the applet (ClearOnDeselect); the initial value is restored once the applet is selected.

Persistent PCRs are reset using the WritePCR APDU.

PCRs are created with any initial value and can be updated by sending data to the PCR; i.e., extend the PCR. PCRs can be reset or deleted, but this is typically protected and not possible for users who create and extend PCRs.

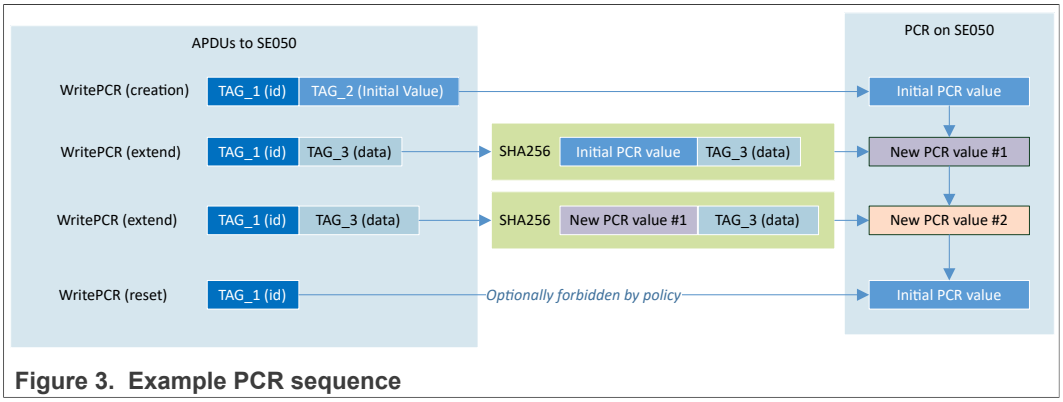

Figure 3. Example PCR sequence

PCRs can be used in object policies to enforce actions, being available only when a PCR value matches the expected value. This can be used, for instance, to enforce an integrity check on a chain of boot loaders.

3.2.1.9 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 3.2.3). 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.

3.2.2 Object types

3.2.2.1 Persistent objects

A persistent Secure Object is fully stored in non-volatile memory, so the content and all the Secure Object attributes are stored persistently.

3.2.2.2 Transient objects

A transient object exists in non-volatile memory, but the transient object content exists in transient memory until the applet is deselected. Therefore, the objects survive a deselect, but the object contents will not survive.

A Secure Object can be constructed to be a transient object by setting the INS_TRANSIENT flag in the INS byte of a C-APDU when creating a Secure Object.

Transient objects can only be used in sessions owned by the user (see Users) who created the object. For example, if user ‘00000001’ creates a transient object, this object can only be used in sessions opened by this user. The same concept applies to the default session. Transient objects created within the default session can only be used in the context of the default session.

If a user tries to use a transient object created by another user, the applet rejects the command with SW ‘6985’. It is also not possible to use importExternalObject for a transient object for the same reason.

The Secure Object attributes are stored persistently.

3.2.3 Authentication object

An Authentication Object is a Secure Object that can only be used to open a session. Sessions cannot be opened by objects that are not Authentication Objects.

Strictly speaking, UserIDs are not Authentication Objects as they do not feature security properties of authentication credentials, but as they also use the session concept of SE05x they are described in this section.

A Secure Object can be constructed to be an Authentication Object by setting a flag in the INS byte of a C-APDU. Authentication objects can only be of class ECKey (only Weierstrass curves can be used for Authentication Objects, others will fail to authenticate), AESKey (only 128 bit) or UserID.

Note that available policies for Authentication Objects are restricted (see Section 3.7.4 for more details).

Once a Secure Object has been created, its size properties cannot be modified; i.e., key bit lengths or file size cannot be increased or decreased. Only persistent Authentication Objects are supported; transient Authentication Objects are not allowed.

Table 2 describes the supported Secure Object types

Table 2. Valid Authentication Object types

Secure Object type	Max. authentication attempt range	Default max. authentication attempts
UserID	[0-255]	Unlimited (= 0)
AESKey (128 bit only)	[0-0x7FFF]	Unlimited (= 0)
ECKey (key pair on Weierstrass curve)	[0-0x7FFF]	Unlimited (= 0)
ECKey (public key on Weierstrass curve)	[0-0x7FFF]	Unlimited (= 0)

3.2.3.1 Users

A user is the entity that opens a session on the secure element.

For the default session, anyone can be the user (as that session is not protected by authentication).

For an authenticated session, the user is defined by the authentication object that is used to authenticate to the SE050. Thus, anyone who knows the content of the authentication object can use it to perform a successful session setup and in that way become a user. There is no distinction on whoever is using the authentication object; the authentication object that is used becomes the reference to the user.

3.2.4 Object attributes

Each Secure Object has a number of attributes assigned to it. These attributes are listed in Table 3.

Table 3. Secure Object Attributes

Attribute	Size (bytes)	Description
Object identifier	4	See Object identifier
Object class	1	See Object class
Authentication indicator	1	See Authentication indicator
Authentication attempts counter	2	See Authentication attempts counter
Authentication Object identifier	4	See Authentication Object identifier
Maximum authentication attempts	2	See Maximum authentication attempts
Policy	Variable	See Policy

Table 3. Secure Object Attributes...*continued*

Attribute	Size (bytes)	Description
Origin	1	See Origin

3.2.4.1 Object identifier

3.2.4.2 Object class

The Object type attribute indicates the class of the Secure Object. See SecureObjectType for the list of supported object types and each associated value.

3.2.4.3 Authentication indicator

The Authentication indicator indicates whether the Secure Object is created as an Authentication Object or not.

The value is one of SetIndicator where SET means the Secure Object is created as Authentication Object and NOT_SET means the Secure Object not created as Authentication Object.

3.2.4.4 Authentication attempts counter

The Authentication attempts counter is a 2-byte value that counts the number of failed authentication attempts.

The counter has an initial value of 0 and will only increase if both:

- the Secure Object is an Authentication Object.
- the Maximum Authentication Attempts has been set to a non-zero value.

Resets to 0 when a successful authentication is performed.

If the Authentication Objects is of type UserID, the authentication attempts are not reported (i.e. the attribute value remains 0).

3.2.4.5 Authentication Object identifier

“Owner” of the secure object; i.e., the 4-byte identifier of the session authentication object when the object has been created.

3.2.4.6 Maximum authentication attempts

Maximum number of authentication attempts.

This 2-byte value can be set when creating a new Secure Object.

The default value is 0, which means unlimited.

3.2.4.7 Policy

Variable length attribute that holds the policy of the Secure Object. See Policies for details.

3.2.4.8 Origin

The Origin attribute is a 1-byte field that indicates the Origin of the Secure Object: either externally set (ORIGIN_EXTERNAL), internally generated (ORIGIN_INTERNAL) or trust provisioned by NXP (ORIGIN_PROVISIONED).

This attribute is updated during applet runtime for Secure Objects of type Key (i.e. AESKey, DESKey, HMACKey, ECKey and RSAKey).

For Secure Objects of type File, the value is always set to ORIGIN_EXTERNAL or ORIGIN_PROVISIONED and will not be updated during applet runtime.

3.2.5 Default configuration

By default, each device will be initialized with the following base configuration:

- EC NIST P-256 curve created and set

Note that the reserved identifiers might have a credential associated (during NXP Trust Provisioning) or not. If no associated credential is present (i.e., the identifier is reserved, but no credential is set), users can create a credential for that particular identifier.

The reserved identifiers are detailed in the next sections.

3.2.5.1 RESERVED_ID_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.

3.2.5.2 RESERVED_ID_ECKEY_SESSION

A device unique key pair which contains the SE050 Key Agreement key pair in ECKey session context.

3.2.5.3 RESERVED_ID_EXTERNAL_IMPORT

A device unique key pair which contains SE050 Key Agreement key pair in ECKey session context; A constant card challenge (all zeroes) is used in order to be able to pre-calculate the encrypted session commands.

3.2.5.4 RESERVED_ID_FEATURE

An authentication object which allows to change the applet variant. This object is created and owned by NXP to define the supported feature set.

3.2.5.5 RESERVED_ID_FACTORY_RESET

An authentication object which allows the user to execute the DeleteAll command which deleted all Secure Objects except objects with Origin set to "ORIGIN_PROVISIONED".

3.2.5.6 RESERVED_ID_UNIQUE_ID

A BinaryFile Secure Object which holds the device unique ID. This file cannot be overwritten or deleted.

3.2.5.7 RESERVED_ID_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 platform SCP03 will be sufficient, not applet session SCP.

3.2.5.8 RESERVED_ID_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.

3.2.5.9 RESERVED_ID_ATTACK_COUNTER

An authentication object which grants access to the attack counter on platform level. This counter stores the value of the current strong attack counter.

3.2.6 Writing Secure Objects

The 4-byte object identifier is used to write the target object. If an object does not yet exist, it will be created. If an object already exists, the value of the object will be updated.

The attributes of an existing object cannot be modified, except the Authentication attempt counter and the Origin (see Table 4).

For any Secure Object op type Key (ECKey, RSAKey, AESKey, DESKey and HMACKey), when the key value is externally generated, the byte size must match exactly the size the expected input size: see Classes for the exact size expected per key type.

Table 4. Secure Object Attribute updatability

Attribute	Updatable after object creation
Object identifier	N
Object type	N
Authentication attribute	N
Authentication attempt counter	Y (only internally from within the applet)
Authentication object identifier	N
Maximum authentication attempts	N
Policy	N
Origin	Y (only applies to Secure Objects of types ECKey, RSAKey, AESKey, DESKey and HMACKey, see Object attributes)

3.2.7 Reading Secure Objects

3.2.7.1 Common read operation

Secure Objects can be read by calling ReadSecureObject function, but only non-secret parts can be returned:

- Reading asymmetric keys will only return the public key.
- Reading symmetric keys will return an error as this is not allowed.

3.2.7.2 Reading with attestation

The user can request attestation for the key or file data requested. Attestation means that the response will have a chip unique identifier, freshness, a timestamp (i.e., monotonic counter value) and a signature over the full payload (requested data, unique identifier, freshness and timestamp) in addition to the requested data.

The signature can only be applied by objects that have the policy POLICY_OBJ_ALLOW_ATTESTATION. If attestation has been requested and the attestation object identifier does not have the policy attached, a SW_SECURITY_STATUS_NOT_SATISFIED will be returned.

Typically, the attestation will occupy the last 4 TLVs in the response in the following order:

1. TLV containing Secure Object attributes (see Object attributes)
2. TLV containing a timestamp value (the timestamp is relative).
3. TLV containing freshness (random data)
4. TLV containing the chip unique ID (see RESERVED_ID_UNIQUE_ID).
5. TLV containing the signature over all the TLV values, including the object attributes, the timestamp, the freshness and the chip unique ID. When applying the signature, all values are concatenated.

If reading with attestation is requested, the user needs to:

- Add INS_ATTEST to the INS byte
- Pass the signing key identifier
- Pass the algorithm used for attestation; the SE050 will perform hashing and signing in case of attestation, both for EC and RSA-based attestation.
- Pass a 16-byte (random) freshness byte array.

Objects can be read with attestation using ReadObject.

Also I2CM response can be read with attestation; see I2CMExecuteCommandSet for details.

3.2.8 Secure Object import/export

Transient 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 key and an 128-bit AES CMAC key. Both keys do not require user interaction, they are internal to the SE050.
- 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.

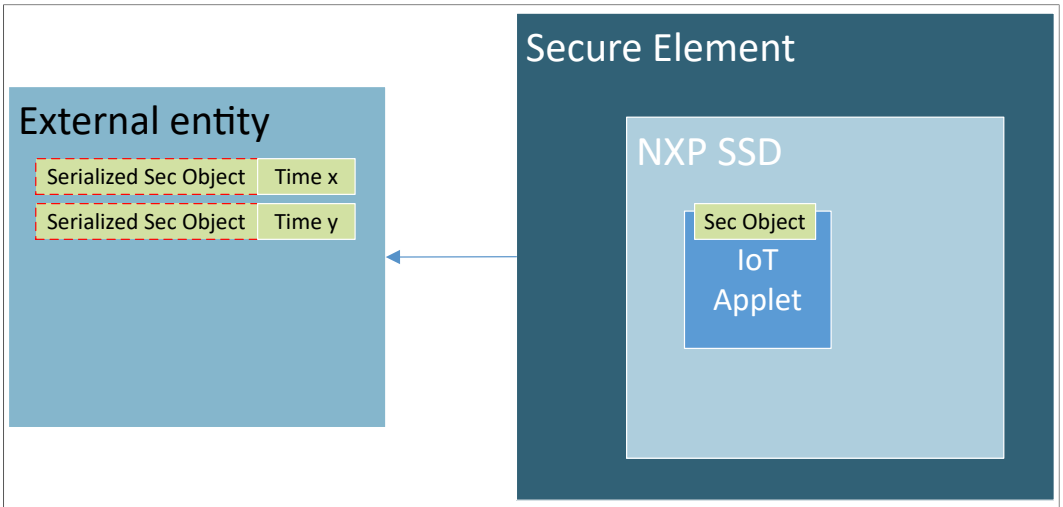

Figure 4. Secure Object import/export

3.2.9 Secure Object external import

Secure Objects can be imported into the SE050 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.

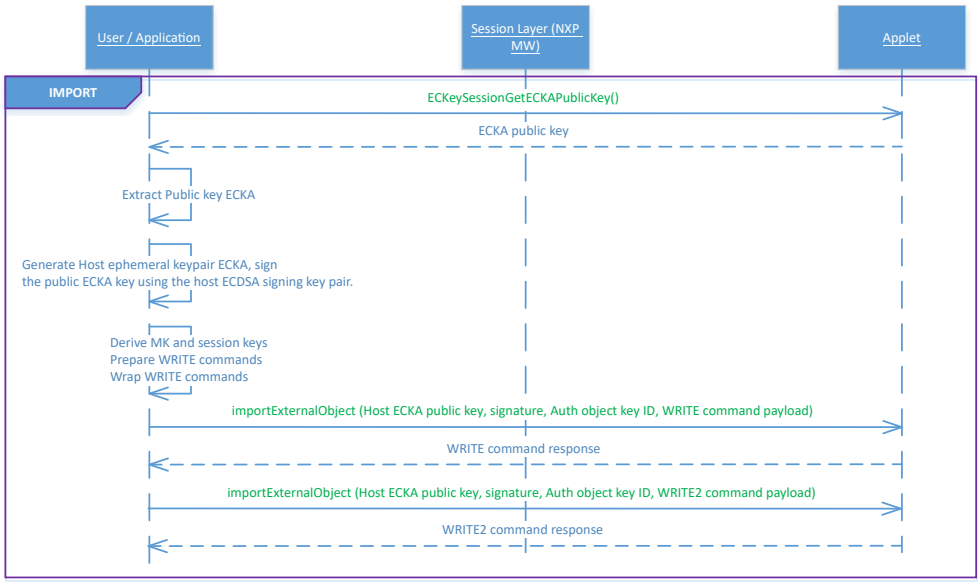

Figure 5. External import flow

The flow above can be summarized in the following steps:

1. The user obtains the SE public key for import via the Section 4.5.4.2 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 Section 4.7.2 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: Table 6).
   - key type, must be 0x88 (AES key type)
   - key length, must be 0x10 (AES128 key)
   - host public key (65-byte NIST P-256 public key)
   - 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 to [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 bytes 00000000000000000000000000000000.
- 2-byte SCP parameter, must equal 0x01 followed by 1-byte security level (which follows the GlobalPlatform security level definition, see: Table 6.
- 1-byte key type

- 1-byte key length

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 SCP03 specification to derive session keys, e.g. derivation input:

- ENC session key = CMAC(MK, 00000000000000000000000400008001)
- CMAC session key = CMAC(MK, 00000000000000000000000600008001)
- RMAC session key = CMAC(MK, 00000000000000000000000700008001)

The Authentication Object ID needs to be passed using TAG_IMPORT_AUTH_KEY_ID, followed by the WriteSecureObject APDU command (using tag TAG_1).

The WriteSecureObject APDU command needs to be constructed as follows:

- Encrypt the command encryption counter (starting with 0x00000000000000000000000000000001) using the ENC session 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>)

The ImportExternalObject commands can only be sent in the default session.

The ImportExternalObject commands are replayable.

See ImportExternalObject for details.

3.3 Crypto Objects

3.3.1 Object types

A Crypto Object is an instance of a Cipher, Digest or Signature that allows users to process data in multiple steps (init/update/final).

The state is lost when the session is closed or expires.

3.3.2 Object identifiers

Crypto Object identifiers are 2 bytes long in the range [0x0000-0xFFFF].

3.3.3 Creating Crypto Objects

The Crypto Object type as well as the Crypto Object sub-type (e.g., Type = Cipher, sub-type = AES_CBC_NOPAD) needs to be specified by the user to create a Crypto Object. The Crypto Object identifier remains available for the user until DeleteCryptoObject APDU command is called.

Note: *The object is created in non-volatile memory and the content remains in transient memory. Also, the creation of a Crypto Context has impact on the available memory, as shown in* Crypto Objects*.*

The following figure shows a flow diagram with an example creation, use and deletion of two Crypto Objects, one used for encrypting a longer data stream and one used for hashing a longer data stream.

Figure 6. Example Crypto Object usage

3.4 Supported applet features

An instance of the SE050 IoT applet can be tuned to support specific functional blocks or features.

There is a bitmap that defines applet features (AppletConfig), which can be set using SetAppletFeatures.

Table 5. Applet features

Feature	Value	Description
CONFIG_ECDAA	0x0001	Use of curve TPM_ECC_BN_P256 (new key creation, key update & crypto operations)
CONFIG_ECDSA_ECDH_ECDHE	0x0002	ECDSA and DH support (new key creation & key update)
CONFIG_EDDSA	0x0004	Use of curve ID_ECC_ED_25519 (new key creation & key update)
CONFIG_DH_MONT	0x0008	Use of curve ID_ECC_MONT_DH_25519 (new key creation & key update).
CONFIG_HMAC	0x0010	Writing HMACKey objects (new key creation & key update)
CONFIG_RSA_PLAIN	0x0020	Writing RSAKey objects (new RSA raw key creation)
CONFIG_RSA_CRT	0x0040	Writing RSAKey objects (new RSA CRT key creation)
CONFIG_AES	0x0080	Writing AESKey objects (new key creation & key update)
CONFIG_DES	0x0100	Writing DESKey objects (new key creation & key update)
CONFIG_PBKDF	0x0200	PBKDF2 (crypto operation)
CONFIG_TLS	0x0400	TLS Handshake support commands (crypto operations, see TLS handshake support)
CONFIG_MIFARE	0x0800	MIFARE DESFire support (crypto operations, see MIFARE DESFire support)
CONFIG_FIPS_MODE_DISABLED	0x1000	If set to 0, the device might operate in FIPS approved mode of operation (see FIPS compliance)
CONFIG_I2CM	0x2000	I2C Master support (crypto operations, see I2C master support)
CONFIG_RESERVED	0x4000	No functionality impact (reserved)
CONFIG_RESERVED	0x8000	No functionality impact (reserved)

3.5 Secure Channel Protocols

3.5.1 Multi-level SCP

The SE050 IoT applet allows the user to set up a secure channel on different levels (i.e., both types are fully independent and can be enabled in parallel):

- **Platform SCP**: for local attack protection. This secure channel needs to be set up via the card manager of the OS using the standard ISO7816-4 secure channel APDUs.
- **Applet level SCP**: for end-to-end secure channel protection. The commands to set up a secure channel on applet level are present in the APDU specification.
  - Users can choose to authenticate with either an AESKey or ECKey to open an AESKey or ECKey session respectively, resulting in session keys that are used for secure messaging on the session.

3.5.2 Security Level

The SE050 IoT applet uses the Security Level definitions as defined in GlobalPlatform, (see Table 10-1 in [SCP03]) and as depicted in Table 6.

Table 6. Security Level

B8	B7	B6	B5	B4	B3	B2	B1	Meaning
1	0	-	-	-	-	-	-	AUTHENTICATED
0	1	-	-	-	-	-	-	ANY_AUTHENTICATED
-	-	-	-	-	-	1	-	C_DECRYPTION
-	-	-	-	-	-	-	1	C_MAC
-	-	1	-	-	-	-	-	R_ENCRYPTION
-	-	-	1	-	-	-	-	R_MAC
-	-	-	-	X	X	-	-	RFU
0	0	0	0	0	0	0	0	NO_SECURITY_LEVEL

3.6 Sessions

The SE050 IoT applet allows to set up **applet sessions**. An applet session is an authenticated communication channel between the owner of an Authentication Object and the SE050 IoT applet.

Commands can be sent to the SE050 IoT applet either:

- Without creating an applet session (= session-less access).
- Inside an applet session.

Each session needs to have a different authentication object; i.e. one Authentication Object cannot be used to open multiple sessions in parallel.

Applet sessions can only be set up via session-less access, so a new applet session cannot be opened from within an existing applet session.

3.6.1 Session-less access

By default, the applet does not require authentication: any command can be sent without creating a session and session-less access is always available (i.e. not closed).

Note that the session-less access does not protect the SE050 use against multi-threaded behavior (as any user or thread can interfere at any moment).

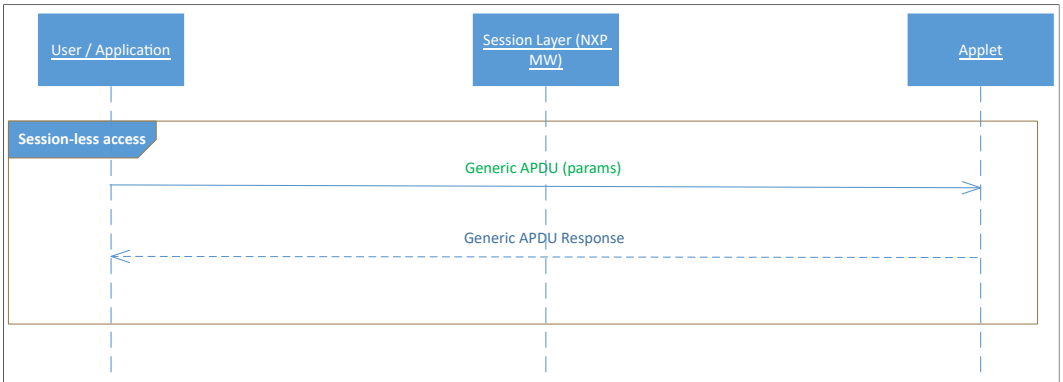

Figure 7. Session-less access

Note:

Without opening an applet session, the APDU prepared by the User / Application are sent directly to the applet

3.6.2 Applet sessions

The following applet session types exist:

- **userID session**: using a userID to open a session
- **AESKey session**: using an AESKey as Authentication Object
- **ECKey session**: using an ECKey as Authentication Object.

To open an (authenticated) applet session, a user must do the following:

1. Call CreateSession, passing an Authentication Object identifier as input and getting an 8-byte unique **session identifier** as response. At this point the session is not yet opened and commands should not be wrapped yet until authentication succeeded.
2. Depending on the type of Authentication Object, authentication needs to occur.
3. Once successfully authenticated, the session is opened. Commands sent within a session are wrapped in a ProcessSessionCmd APDU where the 1st argument is the session identifier and the 2nd argument is the APDU to be handled.

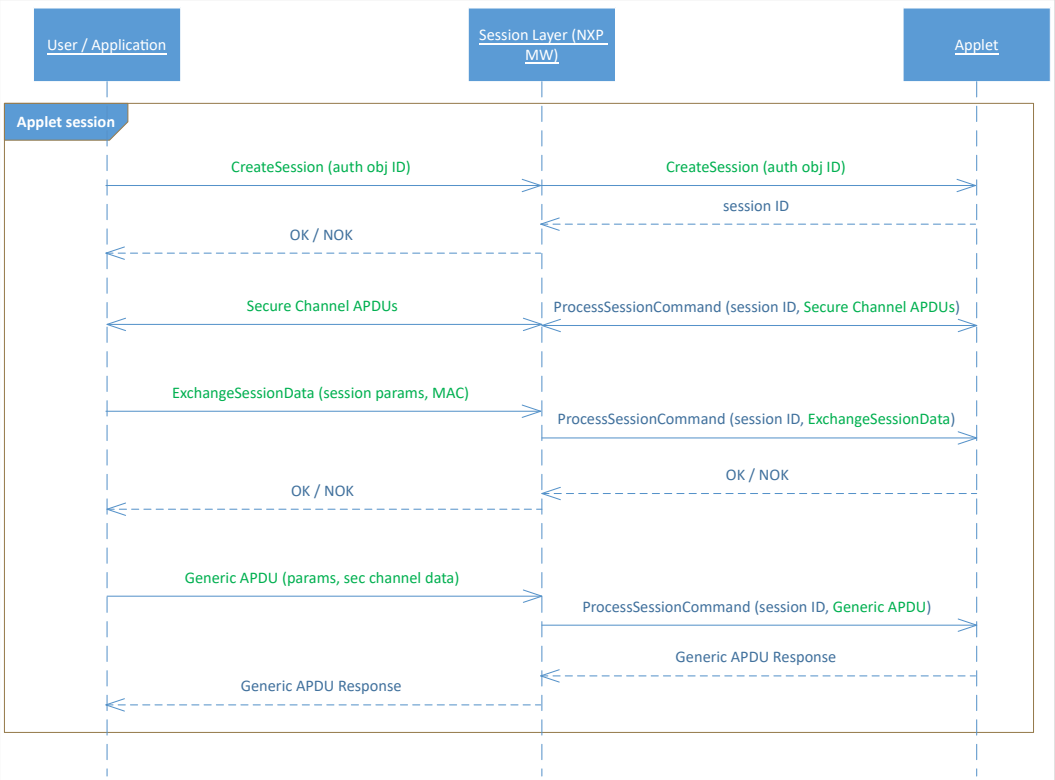

Figure 8. Applet session (overview)

Optionally, the host may provide an ExchangeSessionData command as the first command within a session (see ExchangeSessionData). This command is used to set the policies for the given session. This command shall not be accepted after other commands have been sent within the session.

For example, it is not possible to encrypt data and then set the session policies. In other words, if the user needs to restrict session usage, the first thing to do is to set the policies.

If the ExchangeSessionData command is not provided, the default session policy applies (see Section 3.7.3).

3.6.3 Session creation

As mentioned, the first step is to get a session identifier by calling CreateSession. The Authentication Object identifier will determine the type of session that will be opened, and each session type has different authentication methods associated.

By default MAX NR OF SESSIONS applet sessions can be opened in parallel.

3.6.3.1 UserID session

The session opening is done by providing a previously registered userID:

- VerifySessionUserID passes the value of the userID as argument. If the userID matches the stored value, the session is opened. UserID sessions can only be used or closed once the VerifySessionUserID has returned SW_NO_ERROR.

UserID sessions are only set up once VerifySessionUserID has returned SW_NO_ERROR.

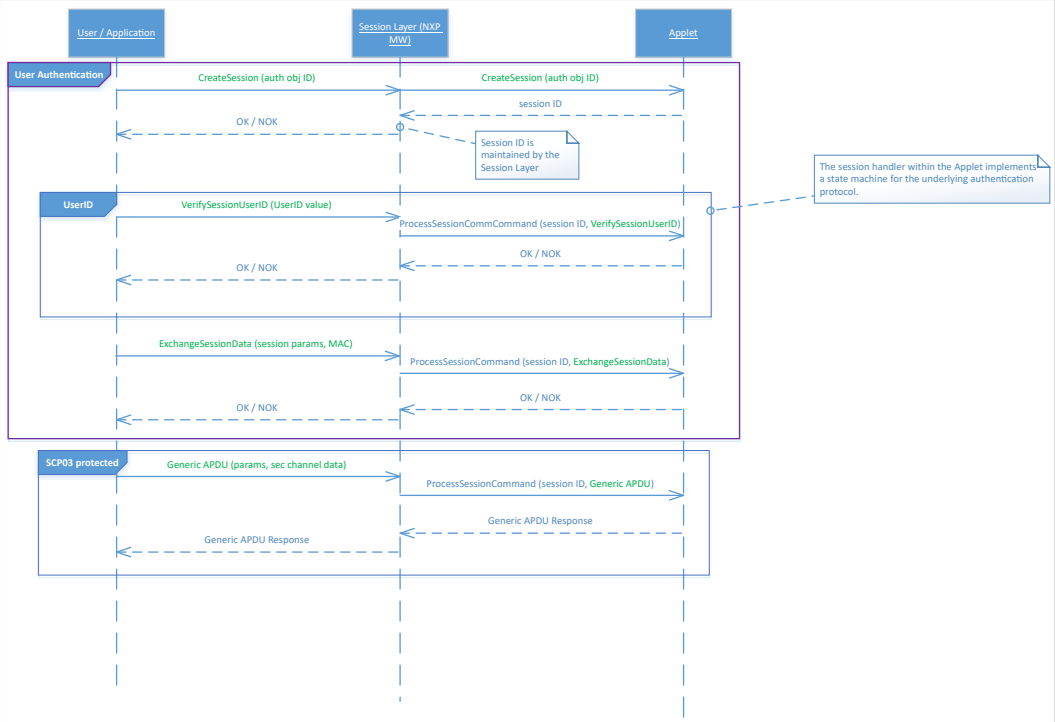

Figure 9. Session creation using UserID

3.6.3.2 AESKey session

Authentication follows the GlobalPlatform authentication steps, namely

1. SCPInitializeUpdate is called to perform an INITIALIZE UPDATE command.
2. SCPExternalAuthenticate is called to perform an EXTERNAL AUTHENTICATE command.

Note that only 1 AESKey object is used as master key for all 3 session keys (S-ENC/S-MAC/S-RMAC); for the derivation input, this master key is used 3 times.

AESKey sessions are only set up once SCPExternalAuthenticate has returned SW_NO_ERROR.

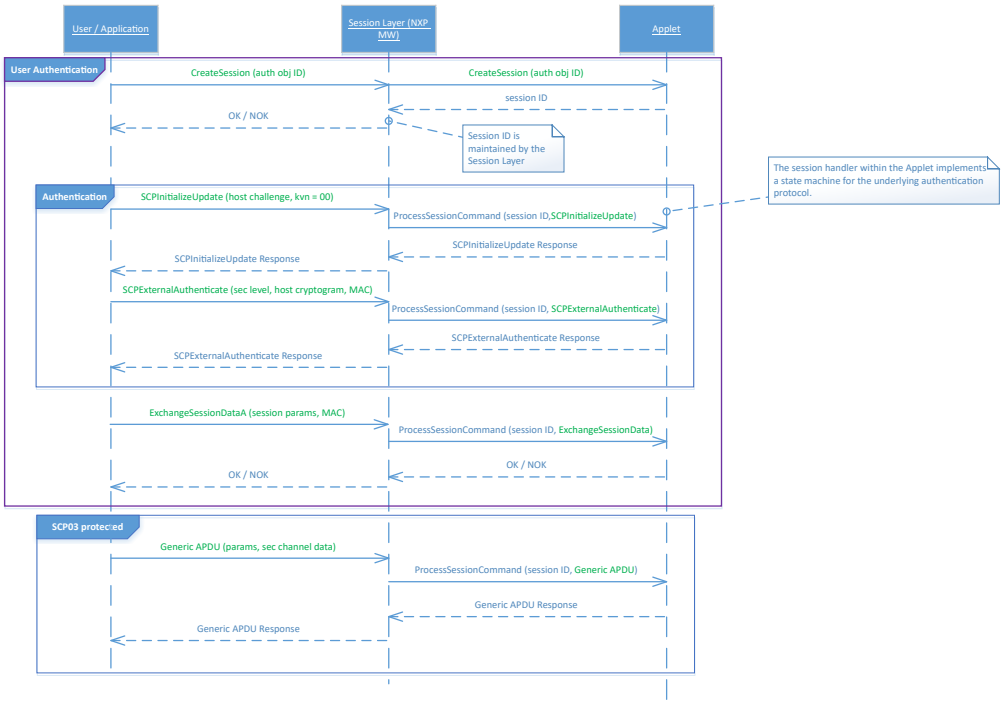

Figure 10. Session creation using an AES key as authentication object

3.6.3.3 ECKey session

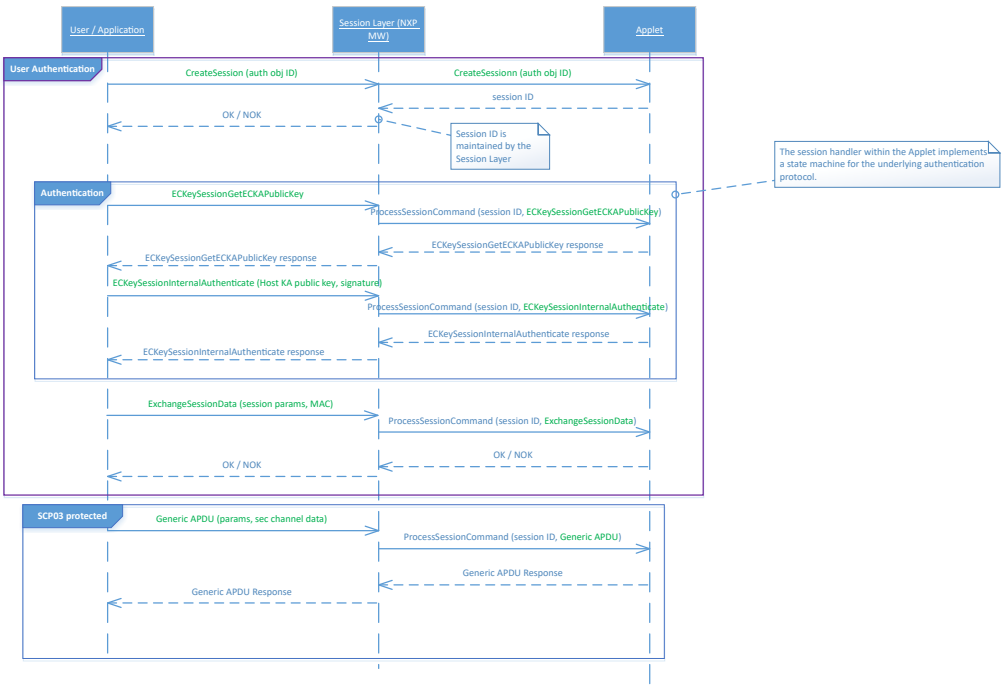

Figure 11. Session creation using ECKey session as authentication mechanism.

Authentication is done as follows:

1.

The user calls ECKeySessionGetECKAPublicKey to get the public key from the device's key pair. Any input key ID except 0x02 will return the public key of the EC key pair with RESERVED_ID_ECKEY_SESSION.

The response is signed by the same key pair.

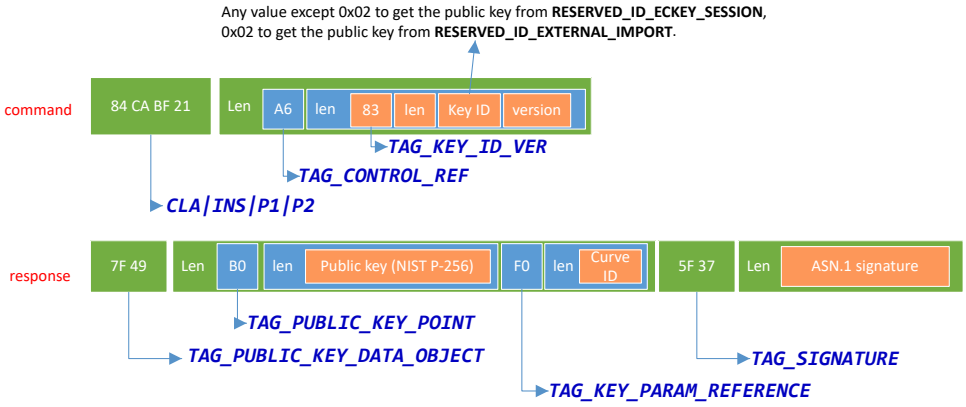

Figure 12. ECKeySessionGetECKAPublicKey

2.

The user calls ECKeySessionInternalAuthenticate with input:

- the applet AID (e.g. A000000396545300000001030000000)
- the SCP parameters
  - 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: Table 6). Note that security level NO_SECURITY_LEVEL is not supported for ECKey sessions.
- key type, must be 0x88 (AES key type)
- key length, must be 0x10 (AES128 key)
- host public key (65-byte NIST P-256 public key)
- host public key curve identifier (must be 0x03 (= NIST_P256))
- ASN.1 signature over the TLV with tags 0xA6 and 0x7F49 (using the S_RMAC key).

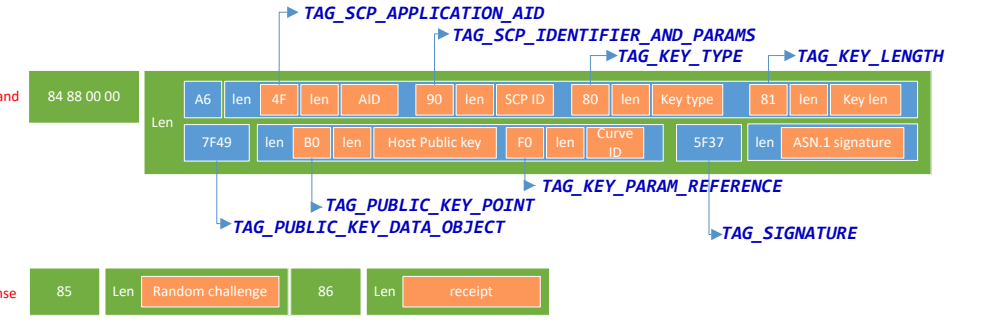

Figure 13. ECKeySessionInternalAuthenticate

The applet will then calculate the master key by performing SHA256 over a byte array containing (in order):

- 4-byte counter value being 0x00000001. Note that this field is totally absent on applets prior to version 3.6.0 (so input to master key calculation is 4 bytes less).
- shared secret (ECDH calculation according to [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 (see Supported EC Curves).
- 16-byte random generated by the SE050.
- 2-byte SCP parameters, parameter, must equal 0x01 followed by 1-byte security level (which follows the GlobalPlatform security level definition, see: security level).
- 1-byte key type
- 1-byte key length

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:

- ENC session key = CMAC(MK, 00000000000000000000000400008001)
- CMAC session key = CMAC(MK, 00000000000000000000000600008001)
- RMAC session key = CMAC(MK, 00000000000000000000000700008001)

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>)

ECKey sessions are only set up once ECKeySessionInternalAuthenticate has returned SW_NO_ERROR.

3.6.4 Session runtime

Sessions can be renewed (by calling RefreshSession from within an existing session).

A refresh means that the session policy is updated with the new policy passed in RefreshSession while the session context remains the same (e.g., state).

When the Authentication Object that is used to open a session is deleted from within that session, the session will be closed automatically immediately after the response APDU has been sent.

When the Authentication Object that is used to open a session is altered, the session remains active.

3.6.5 Session closure

Sessions can be closed in multiple ways:

- explicitly by calling CloseSession
- implicitly due to an applied session policy, i.e. expiry of the session lifetime or reaching the maximum number of allowed APDUs.
- implicitly due to deselect or power cycle.
- implicitly due to deletion of the Authentication Object used to open the session. If the Authentication Object is updated, the session is not closed. If a session is open and the Authentication Object used to open this session is deleted from within another session (using a different Authentication Object), the session remains open until closed in another way.

Sessions are fully transient. If a session expires, its state information is lost.

Note that sessions can only be closed if the session is fully set up; i.e., authentication must be finished successfully.

3.7 Policies

All restrictions that can be applied to Secure Objects or to sessions are constructed as policy sets. A policy set is a combination of different policies on the same item:

- Object policy: defines the restrictions and working conditions of a Secure Object.
- Session policy: defines the restrictions and working conditions of a session.

3.7.1 Object policies

The concepts defined in this section are listed in Table 7

Table 7. Policy notation

Term	Meaning
Policy set	A collection of policies that restrict usage of a specific object; i.e., each object may contain one policy set. An object may also not contain a specific policy set, in which case the default policy set applies.
Policy	A collection of access rules that are applicable to a specific user or a group of users.
Access Rule	Defines the capability to access a resource in a certain manner. For example, an access rule defined within this specification is the capability to use an object for encryption.

3.7.1.1 Policy set

A policy set can be specified when creating an object and it is not modifiable. Policy sets are structured as defined in Table 8.

Table 8. Policy set

Field	Length	Description
Policy	9-45	First policy
Policy	9-45	Second policy
…	…	…

3.7.1.2 Policy

Each policy is structured according to Table 9 and Table 10.

Table 9. Policy

Field	Length	Description	M / O / C
Length of policy	1	Number of bytes of the following fields	M
Authentication Object ID	4	The authentication object to which the following access rules apply. If the value is set to 0x00000000, this means “all other users”. Note that a specific user policy overrides the policy for all other users.	M

Table 9. Policy...*continued*

Field	Length	Description	M / O / C
Access rules (AR)	4-40	See Section 3.7.1.3	M

Table 10. Access Rule structure

	Field	Length	Description	M / O/ C
	AR Header	4	Access rules header of fixed size	M
	AR Extension	0-36	Optional access rules extension	C

3.7.1.3 Access Rule

An access rule defines which operations are allowed on an object. As defined in Table 9 and Table 10 , an access rule contains a mandatory 4-byte header and an optional extension of up to 36 bytes. The coding of the header and extensions is defined in section Policy Constants.

Access rule extensions are conditional to the presence of specific access rules. If an access rules requires extension, then the extension shall be present; otherwise the access rule shall be deemed invalid. Extensions are added from left to right in the same order in which the access rules are defined. As an example, consider that a specific object defines a policy for an Authentication Object ID (e.g., identifier = '7FFF0000') as follows:

- Read access is granted (POLICY_OBJ_ALLOW_READ)
- A PCR object with ID '4FFFF0000' shall have value '00112233445566778899AABBCCDDEEFF00112233445566778899AABBCCDDEEFF' (POLICY_OBJ_REQUIRE_PCR_VALUE)

The above example policy would be coded as follows:

- Policy length: '28' (40 bytes total)
- Access rule header: '00201000' (POLICY_OBJ_ALLOW_READ | POLICY_OBJ_REQUIRE_PCR_VALUE)
- Access rule extension: '4FFFF00000011223344556677889 9AABBCCDDEEFF 00112233445566778899AABBCCDDEEFF'

3.7.1.4 Policy validation

Policies are validated during the object creation. An object is only created if the attached policy is valid and, if the policy validation fails, the error code 0x6A80 is returned as response to the object creation command.

Besides checking the policy structure and length, the following rules are checked:

- If no policy is attached, the default policies are applied, and no more checks are performed;
- Each access rule is checked against the object type. For example, a symmetric key shall not contain the policy POLICY_OBJ_ALLOW_READ;

Table 11 defines which access rules are allowed for each object class, as defined in Classes.

Table 11. Policy validation per object type

Object class	Applicable access rules
All classes (policies applicable to all classes defined below, regardless of their type)	POLICY_OBJ_ALLOW_DELETE POLICY_OBJ_REQUIRE_SM POLICY_OBJ_REQUIRE_PCR_VALUE POLICY_OBJ_FORBID_ALL
Symmetric key (AES, DES, HMAC)	POLICY_OBJ_ALLOW_SIGN POLICY_OBJ_ALLOW_VERIFY POLICY_OBJ_ALLOW_ENC (does not apply for HMACKey Secure Objects) POLICY_OBJ_ALLOW_DEC (does not apply for HMACKey Secure Objects) POLICY_OBJ_ALLOW_KDF (only applies to HMACKey Secure Objects) POLICY_OBJ_ALLOW_WRAP POLICY_OBJ_ALLOW_WRITE POLICY_OBJ_ALLOW_GEN POLICY_OBJ_ALLOW_DESFIRE_AUTHENTICATION (only applies to AESKey Secure Objects) POLICY_OBJ_ALLOW_DESFIRE_DUMP_SESSION_KEYS (only applies to AESKey Secure Objects) POLICY_OBJ_ALLOW_IMPORT_EXPORT
Asymmetric key (RSA, EC)	POLICY_OBJ_ALLOW_SIGN POLICY_OBJ_ALLOW_VERIFY POLICY_OBJ_ALLOW_ENC POLICY_OBJ_ALLOW_DEC POLICY_OBJ_ALLOW_KDF POLICY_OBJ_ALLOW_WRAP POLICY_OBJ_ALLOW_WRITE POLICY_OBJ_ALLOW_GEN POLICY_OBJ_ALLOW_KA POLICY_OBJ_ALLOW_READ POLICY_OBJ_ALLOW_ATTESTATION POLICY_OBJ_ALLOW_IMPORT_EXPORT
Binary File, Counter, PCR	POLICY_OBJ_ALLOW_WRITE POLICY_OBJ_ALLOW_READ
UserID	POLICY_OBJ_ALLOW_WRITE

3.7.2 Session policies

A policy may be associated to a session while opening a session. A policy controls certain aspects of session lifecycle.

Session policies are structured as per Table 12.

Table 12. Session policy

Field	Length	Description
Length of policy	1	The number of bytes of the policy (a value between 2 and 6)
Header	2	Bitmap encoding access rules for a session

Table 12. Session policy...*continued*

Field	Length	Description
Extension	0-4	Optional extension

The extension bytes are optional and follow the same rules as defined for object policies. The policies applicable to sessions are detailed in section Session policy.

3.7.3 Default policies

This section defines the default policy rules per object type.

Table 13. Default policies

Object type	Default policy
Authentication Object	Maximum attempts: unlimited. Applied policies: POLICY_OBJ_ALLOW_READ
Generic SecureObject (non-authentication object)	Applied policies: All policies as depicted in Table 11, except POLICY_OBJ_REQUIRE_SM, POLICY_OBJ_REQUIRE_PCR_VALUE, POLICY_OBJ_FORBID_ALL and POLICY_OBJ_ALLOW_ATTESTATION.
Session	No maximum number of APDU or command limitations Session refresh is not allowed

3.7.4 Authentication Object policies

Authentication objects are limited to the following policies or a subset thereof:

- POLICY_OBJ_ALLOW_READ
- POLICY_OBJ_ALLOW_WRITE
- POLICY_OBJ_ALLOW_GEN
- POLICY_OBJ_ALLOW_DELETE
- POLICY_OBJ_REQUIRE_SM
- POLICY_OBJ_REQUIRE_PCR_VALUE
- POLICY_OBJ_FORBID_ALL

There is only one global setting that is specific for authentication objects. That is the maximum number of attempts to authenticate using the object. This is a setting passed in the input data of the command to create the authentication object. The value of the maximum attempts is coded in 2 bytes of TAG_MAX_ATTEMPTS.

3.8 Lifecycle management

The applet has 2 different lifecycle states:

- Active – all commands are allowed (as long as they do not violate policies)
- Inactive – only a subset of commands is allowed.

Commands that are allowed in Inactive state are defined in Table 14.

Table 14. Commands allowed in Inactive state

Command	Remark
GetVersion	

Table 14. Commands allowed in Inactive state...*continued*

Command	Remark
ReadObject	Only object with identifier Section 3.2.5.6 can be read.
GetRandom	
CreateSession	

The applet can be set to Inactive state calling SetLockState.

Unlocking the applet can only be done by a successful authentication using the reserved authentication object with identifier RESERVED_ID_TRANSPORT.

3.9 Timestamp functionality

The system provides timestamps during attestation. A timestamp is a relative counter value of 12 bytes of which the most significant 4 bytes are persistent and the least significant 8 bytes are transient.

The transient part is updated at least every 100 msec.

The persistent part is updated on each first call to get an attested read or the first call to GetTimestamp.

3.10 FIPS compliance

The SE050 runs in FIPS 140-2, Level 3 (Physical security at Level 4) “approved mode of operation” only if the applet feature set is restricted to the following features being set to 1:

- CONFIG_ECDSA_ECDH_ECDHE
- CONFIG_HMAC
- CONFIG_DES
- CONFIG_AES
- CONFIG_I2CM
- CONFIG_RSA_CRT
- CONFIG_RESERVED (these can be set or unset => no influence to FIPS approved mode of operation)

All other applet features as mentioned in Supported Applet Features must be set to 0.

When CONFIG_MODE_FIPS_DISABLED is not set, the function ECDHGenerateShared secret will always return SW_CONDITIONS_NOT_SATISFIED.

When CONFIG_MODE_FIPS_DISABLED is not set, at least the following Secure Objects are trust provisioned:

- RESERVED_ID_ECKEY_SESSION
- RESERVED_ID_EXTERNAL_IMPORT
- RESERVED_ID_FEATURE
- RESERVED_ID_PLATFORM_SCP

4 SE050 APDU interface

4.1 APDU Format

SE050 IoT applet defines APDUs according to [ISO7816-4] APDU message format. Both standard as well as extended length APDUs are supported. APDUs described in the document use standard length APDU format notation, but extended length APDUs are supported as well.

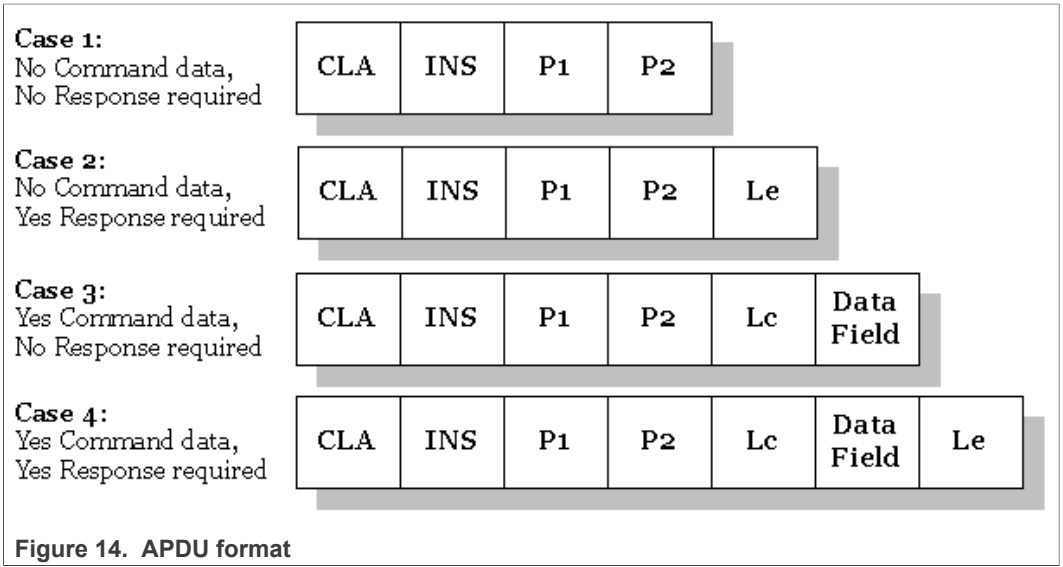

Figure 14. APDU format

4.1.1 APDU header

4.1.1.1 CLA byte

The CLA byte is fixed for each command to 0x80 (= no secure messaging) or 0x84 (= proprietary secure messaging). Any other CLA byte will be rejected.

If APDUs are wrapped (as payload to ProcessSessionCmd), the CLA byte of the wrapper is checked, but the CLA byte of the APDU command in the payload is not checked.

4.1.2 Le field

No explicit checks are done on the Le field validity by the applet. Le field must in any case be smaller than 0x8000.

4.1.3 TLV based payloads

All APDU's have TLV based payload according to [ISO7816-4] Annex D with some exceptions, as mentioned below.

4.1.3.1 TLV Tag encoding

The specification allows 1-byte Tags only; any value 0x00 up to 0xFF is possible, so this does not comply to [ISO7816-4] Annex D.2: Tag field

4.1.3.2 TLV Length encoding

According [ISO7816-4] Annex D.3: Length field

The length field is limited to 3 bytes maximum (in that case 0x82 followed by 2 bytes indicating the length).

R-APDUs might use a 3-byte L field, even if the length is less than 128 bytes.

4.1.3.3 TLV Value encoding

According [ISO7816-4] Annex D.4: Value field

4.1.4 TLV description

Each TLV will be described with one of the following descriptions:

- *[Optional]* means that the TLV can be used or not; up to the user to decide.
- *[Conditional: <condition>; <error code>]* will indicate that the TLV is conditional where <condition> specifies the condition which is applicable and <error code> specifies the expected error code in case the condition is not fulfilled; e.g.:
  - [Conditional: object does not yet exist; SW_WRONG_DATA] would mean that the TLV is needed when the object does not yet exist. If the TLV is absent in that case, the returned error code would be SW_WRONG_DATA.
  - Note that the error code is not always present. In that case any error code should be assumed.
- If neither [Optional] nor [Conditional] are mentioned, then the TLV is [Mandatory].

A TLV can be Optional and Conditional at the same time. Then the Condition must apply and it is then up to the user to use the TLV or not.

Note that for some APDUs, certain TLVs might be skipped, so it could be an APDU uses e.g., TLV[TAG_1], TLV[TAG_2], TLV[TAG_4], but not TLV[TAG_3].

4.1.5 TLV order

TLVs described for C-APDU must always come in the order as described for an APDU, so users cannot mix the order of TLVs in the C-APDU payload.

4.2 Error codes

Each APDU will list a number of error codes. Note that the listed error codes on each APDU are not limiting; i.e., if another error code is returned, it means the APDU has failed processing and users should take care of appropriate error handling.

4.3 Constants

4.3.1 Error codes

Table 15. Error codes

Name	Value	Description
SW_NO_ERROR	0x9000	No Error
SW_CONDITIONS_NOT_SATISFIED	0x6985	Conditions not satisfied

Table 15. Error codes*...continued*

Name	Value	Description
SW_SECURITY_STATUS	0x6982	Security status not satisfied.
SW_WRONG_DATA	0x6A80	Wrong data provided.
SW_DATA_INVALID	0x6984	Data invalid – policy set invalid for the given object
SW_COMMAND_NOT_ALLOWED	0x6986	Command not allowed – access denied based on object policy

4.3.2 General

Table 16. General constants

Name	Value	Description
MAX_NUMBER_OF_SESSIONS	2	Maximum number of simultaneous applet sessions (excl. session-less access)
OBJECT_IDENTIFIER_SIZE	4	
CRYPTO_OBJECT_IDENTIFIER_SIZE	2	
MAX_I2CM_COMMAND_LENGTH	255	
MAX_APDU_PAYLOAD_LENGTH	889	The APDU buffer is 894 bytes in total, the maximum payload length of 889 bytes only applies when no secure messaging is applied. Or the maximum APDU payload length will be smaller, depending on which protocol applies, etc.

4.3.3 Instruction

Table 17. Instruction mask constants

Name	Value	Description
INS_MASK_INS_CHAR	0xE0	3 MSBit for instruction characteristics.
INS_MASK_INSTRUCTION	0x1F	5 LSBit for instruction

Table 18. Instruction characteristics constants

Name	Value	Description
INS_TRANSIENT	0x80	Mask for transient object creation, can only be combined with INS_WRITE. This bit is ignored when the Secure Object already exists.
INS_AUTH_OBJECT	0x40	Mask for authentication object creation, can only be combined with INS_WRITE. This bit is ignored when the Secure Object already exists.
INS_ATTEST	0x20	Mask for getting attestation data.

Table 19. Instruction constants

Name	Value	Description
INS_WRITE	0x01	Write or create a persistent object.
INS_READ	0x02	Read the object
INS_CRYPTO	0x03	Perform Security Operation
INS_MGMT	0x04	General operation
INS_PROCESS	0x05	Process session command
INS_IMPORT_EXTERNAL	0x06	Import external object

4.3.4 P1 parameter

Table 20. P1Mask constants

Name	Value	Description
P1_UNUSED	0x80	Highest bit not used
P1_MASK_KEY_TYPE	0x60	2 MSBit for key type
P1_MASK_CRED_TYPE	0x1F	5 LSBit for credential type

Table 21. P1KeyType constants

Name	Value	Description
P1_KEY_PAIR	0x60	Key pair (private key + public key)
P1_PRIVATE	0x40	Private key
P1_PUBLIC	0x20	Public key

Table 22. P1Cred constants

Name	Value
P1_DEFAULT	0x00
P1_EC	0x01
P1_RSA	0x02
P1_AES	0x03
P1_DES	0x04
P1_HMAC	0x05
P1_BINARY	0x06
P1_USERID	0x07
P1_COUNTER	0x08
P1_PCR	0x09
P1_CURVE	0x0B
P1_SIGNATURE	0x0C
P1_MAC	0x0D

Table 22. P1Cred constants*...continued*

Name	Value
P1_CIPHER	0x0E
P1_TLS	0x0F
P1_CRYPTO_OBJ	0x10

4.3.5 P2 parameter

Table 23. P2 constants

Name	Value
P2_DEFAULT	0x00
P2_GENERATE	0x03
P2_CREATE	0x04
P2_SIZE	0x07
P2_SIGN	0x09
P2_VERIFY	0x0A
P2_INIT	0x0B
P2_UPDATE	0x0C
P2_FINAL	0x0D
P2_ONESHOT	0x0E
P2_DH	0x0F
P2_DIVERSIFY	0x10
P2_AUTH_FIRST_PART2	0x12
P2_AUTH_NONFIRST_PART2	0x13
P2_DUMP_KEY	0x14
P2_CHANGE_KEY_PART1	0x15
P2_CHANGE_KEY_PART2	0x16
P2_KILL_AUTH	0x17
P2_IMPORT	0x18
P2_EXPORT	0x19
P2_SESSION_CREATE	0x1B
P2_SESSION_CLOSE	0x1C
P2_SESSION_REFRESH	0x1E
P2_SESSION_POLICY	0x1F
P2_VERSION	0x20
P2_MEMORY	0x22
P2_LIST	0x25
P2_TYPE	0x26
P2_EXIST	0x27

Table 23. P2 constants*...continued*

Name	Value
P2_DELETE_OBJECT	0x28
P2_DELETE_ALL	0x2A
P2_SESSION_USERID	0x2C
P2_HKDF	0x2D
P2_PBKDF	0x2E
P2_I2CM	0x30
P2_I2CM_ATTESTED	0x31
P2_MAC	0x32
P2_UNLOCK_CHALLENGE	0x33
P2_CURVE_LIST	0x34
P2_SIGN_ECDAA	0x35
P2_ID	0x36
P2_ENCRYPT_ONESHOT	0x37
P2_DECRYPT_ONESHOT	0x38
P2_ATTEST	0x3A
P2_ATTRIBUTES	0x3B
P2_CPLC	0x3C
P2_TIME	0x3D
P2_TRANSPORT	0x3E
P2_VARIANT	0x3F
P2_PARAM	0x40
P2_DELETE_CURVE	0x41
P2_ENCRYPT	0x42
P2_DECRYPT	0x43
P2_VALIDATE	0x44
P2_GENERATE_ONESHOT	0x45
P2_VALIDATE_ONESHOT	0x46
P2_CRYPTO_LIST	0x47
P2_RANDOM	0x49
P2_TLS_PMS	0x4A
P2_TLS_PRF_CLI_HELLO	0x4B
P2_TLS_PRF_SRV_HELLO	0x4C
P2_TLS_PRF_CLI_RND	0x4D
P2_TLS_PRF_SRV_RND	0x4E
P2_RAW	0x4F
P2_IMPORT_EXT	0x51

Table 23. P2 constants*...continued*

Name	Value
P2_SCP	0x52
P2_AUTH_FIRST_PART1	0x53
P2_AUTH_NONFIRST_PART1	0x54

4.3.6 SecureObject type

Table 24. SecureObjectType constants

Name	Value
TYPE_EC_KEY_PAIR	0x01
TYPE_EC_PRIV_KEY	0x02
TYPE_EC_PUB_KEY	0x03
TYPE_RSA_KEY_PAIR	0x04
TYPE_RSA_KEY_PAIR_CRT	0x05
TYPE_RSA_PRIV_KEY	0x06
TYPE_RSA_PRIV_KEY_CRT	0x07
TYPE_RSA_PUB_KEY	0x08
TYPE_AES_KEY	0x09
TYPE_DES_KEY	0x0A
TYPE_BINARY_FILE	0x0B
TYPE_USERID	0x0C
TYPE_COUNTER	0x0D
TYPE_PCR	0x0F
TYPE_CURVE	0x10
TYPE_HMAC_KEY	0x11

4.3.7 Memory

Table 25. Memory constants

Name	Value	Description
MEM_PERSISTENT	0x01	Persistent memory
MEM_TRANSIENT_RESET	0x02	Transient memory, clear on reset
MEM_TRANSIENT_DESELECT	0x03	Transient memory, clear on deselect

4.3.8 Origin

Table 26. Origin constants

Name	Value	Description
ORIGIN_EXTERNAL	0x01	Generated outside the module.
ORIGIN_INTERNAL	0x02	Generated inside the module.
ORIGIN_PROVISIONED	0x03	Trust provisioned by NXP

4.3.9 TLV tags

Table 27. Tags

Name	Value
TAG_SESSION_ID	0x10
TAG_POLICY	0x11
TAG_MAX_ATTEMPTS	0x12
TAG_IMPORT_AUTH_DATA	0x13
TAG_IMPORT_AUTH_KEY_ID	0x14
TAG_1	0x41
TAG_2	0x42
TAG_3	0x43
TAG_4	0x44
TAG_5	0x45
TAG_6	0x46
TAG_7	0x47
TAG_8	0x48
TAG_9	0x49
TAG_10	0x4A

4.3.10 ECSignatureAlgo

Table 28. ECSignatureAlgo

Name	Value	Description
SIG_ECDSA_PLAIN	0x09	NOT SUPPORTED
SIG_ECDSA_SHA	0x11	ECDSA with a SHA-1 digest as input.
SIG_ECDSA_SHA_224	0x25	ECDSA with a SHA224 digest as input.
SIG_ECDSA_SHA_256	0x21	ECDSA with a SHA256 digest as input.
SIG_ECDSA_SHA_384	0x22	ECDSA with a SHA384 digest as input.
SIG_ECDSA_SHA_512	0x26	ECDSA with a SHA512 digest as input.

4.3.11 EDSignatureAlgo

Table 29. EDSignatureAlgo

Name	Value	Description
SIG_ED25519PURE	0xA3	EDDSA Pure (using SHA512 as digest)

4.3.12 ECDAASignatureAlgo

Table 30. ECDAASignatureAlgo

Name	Value	Description
SIG_ECDAA	0xF4	Message input must be pre-hashed (using SHA256)

4.3.13 RSASignatureAlgo

Table 31. RSASignatureAlgo

Name	Value	Description
RSA_SHA1_PKCS1_PSS	0x15	RFC8017: RSASSA-PSS
RSA_SHA224_PKCS1_PSS	0x2B	RFC8017: RSASSA-PSS
RSA_SHA256_PKCS1_PSS	0x2C	RFC8017: RSASSA-PSS
RSA_SHA384_PKCS1_PSS	0x2D	RFC8017: RSASSA-PSS
RSA_SHA512_PKCS1_PSS	0x2E	RFC8017: RSASSA-PSS
RSA_SHA1_PKCS1	0x0A	RFC8017: RSASSA-PKCS1-v1_5
RSA_SHA_224_PKCS1	0x27	RFC8017: RSASSA-PKCS1-v1_5
RSA_SHA_256_PKCS1	0x28	RFC8017: RSASSA-PKCS1-v1_5
RSA_SHA_384_PKCS1	0x29	RFC8017: RSASSA-PKCS1-v1_5
RSA_SHA_512_PKCS1	0x2A	RFC8017: RSASSA-PKCS1-v1_5

4.3.14 RSAEncryptionAlgo

Table 32. RSAEncryptionAlgo

Name	Value	Description
RSA_NO_PAD	0x0C	Plain RSA, padding required on host.
RSA_PKCS1	0x0A	RFC8017: RSAES-PKCS1-v1_5
RSA_PKCS1_OAEP	0x0F	RFC8017: RSAES-OAEP (using SHA1 as digest)

4.3.15 RSABitLength

Table 33. RSABitLength

Name	Value
RSA_512	512
RSA_1024	1024
RSA_1152	1152
RSA_2048	2048
RSA_3072	3072
RSA_4096	4096

4.3.16 RSAKeyComponent

Table 34. RSAKeyComponent

Name	Value	Description
RSA_COMP_MOD	0x00	Modulus
RSA_COMP_PUB_EXP	0x01	Public key exponent
RSA_COMP_PRIV_EXP	0x02	Private key exponent
RSA_COMP_P	0x03	CRT component p
RSA_COMP_Q	0x04	CRT component q
RSA_COMP_DP	0x05	CRT component dp
RSA_COMP_DQ	0x06	CRT component dq
RSA_COMP_INVQ	0x07	CRT component q_inv

4.3.17 DigestMode

Table 35. DigestMode constants

Name	Value
DIGEST_NO_HASH	0x00
DIGEST_SHA	0x01
DIGEST_SHA224	0x07
DIGEST_SHA256	0x04
DIGEST_SHA384	0x05
DIGEST_SHA512	0x06

4.3.18 MACAlgo

Table 36. MACAlgo constants

Name	Value
HMAC_SHA1	0x18
HMAC_SHA256	0x19
HMAC_SHA384	0x1A
HMAC_SHA512	0x1B
CMAC_128 (ISO9797 M2 padding)	0x31
DES_MAC4_ISO9797_M2	0x05
DES_MAC4_ISO9797_1_M2_ALG3	0x13
DES_MAC4_ISO9797_M1	0x03
DES_MAC4_ISO9797_1_M1_ALG3	0x2F
DES_MAC8_ISO9797_M2	0x06
DES_MAC8_ISO9797_1_M2_ALG3	0x14
DES_MAC8_ISO9797_1_M1_ALG3	0x04
DES_MAC8_ISO9797_1_M1_ALG3	0x30

Table 36. MACAlgo constants...*continued*

Name	Value
CMAC128	0x31
DES_CMAC8	0x7A
AES_CMAC16	0x66

4.3.19 ECCurve

Table 37. ECCurve constants

Name	Curve ID	Weierstrass
UNUSED	0x00	-
NIST_P192	0x01	Y
NIST_P224	0x02	Y
NIST_P256	0x03	Y
NIST_P384	0x04	Y
NIST_P521	0x05	Y
Brainpool160	0x06	Y
Brainpool192	0x07	Y
Brainpool224	0x08	Y
Brainpool256	0x09	Y
Brainpool320	0x0A	Y
Brainpool384	0x0B	Y
Brainpool512	0x0C	Y
Secp160k1	0x0D	Y
Secp192k1	0x0E	Y
Secp224k1	0x0F	Y
Secp256k1	0x10	Y
TPM_ECC_BN_P256 (Barreto-Naehrig curve)	0x11	Y
ID_ECC_ED_25519	0x40	N
ID_ECC_MONT_DH_25519	0x41	N

4.3.20 ECCurveParam

Table 38. ECCurveParam constants

Name	Value
CURVE_PARAM_A	0x01
CURVE_PARAM_B	0x02
CURVE_PARAM_G	0x04
CURVE_PARAM_N	0x08
CURVE_PARAM_PRIME	0x10

4.3.21 CipherMode

Table 39. CipherMode constants

Name	Value	Description
DES_CBC_NOPAD	0x01	Typically using DESKey identifiers
DES_CBC_ISO9797_M1	0x02	Typically using DESKey identifiers
DES_CBC_ISO9797_M2	0x03	Typically using DESKey identifiers
DES_CBC_PKCS5	0x04	NOT SUPPORTED
DES_ECB_NOPAD	0x05	Typically using DESKey identifiers
DES_ECB_ISO9797_M1	0x06	NOT SUPPORTED
DES_ECB_ISO9797_M2	0x07	NOT SUPPORTED
DES_ECB_PKCS5	0x08	NOT SUPPORTED
AES_ECB_NOPAD	0x0E	Typically using AESKey identifiers
AES_CBC_NOPAD	0x0D	Typically using AESKey identifiers
AES_CBC_ISO9797_M1	0x16	Typically using AESKey identifiers
AES_CBC_ISO9797_M2	0x17	Typically using AESKey identifiers
AES_CBC_PKCS5	0x18	NOT SUPPORTED
AES_CTR	0xF0	Typically using AESKey identifiers

4.3.22 AttestationAlgo

AttestationAlgo is either ECSignatureAlgo or RSASignatureAlgo.

4.3.23 AppletConfig

Table 40. Applet configurations

Name	Value
CONFIG_ECDAA	0x0001
CONFIG_ECDSA_ECDH_ECDHE	0x0002
CONFIG_EDDSA	0x0004
CONFIG_DH_MONT	0x0008
CONFIG_HMAC	0x0010
CONFIG_RSA_PLAIN	0x0020
CONFIG_RSA_CRT	0x0040
CONFIG_AES	0x0080
CONFIG_DES	0x0100
CONFIG_PBKDF	0x0200
CONFIG_TLS	0x0400
CONFIG_MIFARE	0x0800
CONFIG_FIPS_MODE_DISABLED	0x1000
CONFIG_I2CM	0x2000

Table 40. Applet configurations*...continued*

Name	Value
CONFIG_ECC_ALL	0x000F
CONFIG_RSA_ALL	0x0060
CONFIG_ALL	0x3FFF

4.3.24 LockIndicator

Table 41. LockIndicator constants

Name	Value
TRANSIENT_LOCK	0x01
PERSISTENT_LOCK	0x02

4.3.25 LockState

Table 42. LockState constants

Name	Value
LOCKED	0x01
UNLOCKED	Any except 0x01

4.3.26 CryptoContext

Table 43. CryptoContext constants

Name	Value	Description
CC_DIGEST	0x01	For DigestInit/DigestUpdate/DigestFinal
CC_CIPHER	0x02	For CipherInit/CipherUpdate/CipherFinal
CC_SIGNATURE	0x03	For MACInit/MACUpdate/MACFinal

4.3.27 Result

Table 44. Result constants

Name	Value
RESULT_SUCCESS	0x01
RESULT_FAILURE	0x02

4.3.28 TransientIndicator

Table 45. TransientIndicator constants

Name	Value
PERSISTENT	0x01
TRANSIENT	0x02

4.3.29 SetIndicator

Table 46. SetIndicator constants

Name	Value
NOT_SET	0x01
SET	0x02

4.3.30 MoreIndicator

Table 47. MoreIndicator constants

Name	Value	Description
NO_MORE	0x01	No more data available
MORE	0x02	More data available

4.3.31 PlatformSCPRequest

Table 48. PlatformSCPRequest constants

Name	Value	Description
SCP_REQUIRED	0x01	Platform SCP is required (full enc & MAC)
SCP_NOT_REQUIRED	0x02	No platform SCP required.

4.3.32 CryptoObject

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

4.3.33 VersionInfo

VersionInfo is a 7-byte value consisting of:

- 1-byte Major applet version
- 1-byte Minor applet version
- 1-byte patch applet version
- 2-byte AppletConfig, indicating the supported applet features
- 2-byte Secure Box version: major version (MSB) concatenated with minor version (LSB).

4.3.34 Policy constants

A notation will be used to identify specific bits: the most significant Byte is 1 and the most significant bit is 8; so if B2b7 is set, this would be coded as 0x00 0x40.

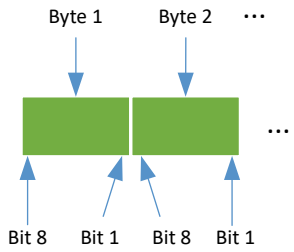

Figure 15. Policy notation

4.3.34.1 Session policy

The session policy header is coded as follows:

Table 49. Session policies

Policy name	Description	Position in Header	Extension required?	Extension length
POLICY_SESSION_MAX_APDU	Defines the maximum number of APDUs allowed within the session. Note that the ExchangeSessionData command itself is also counted as APDU within the session.	B1b8	Y	2
RFU		B1b7	n/a	
POLICY_SESSION_ALLOW_REFRESH	Defines whether this session can be refreshed without losing context.	B1b6	N	
RFU	Other values reserved for future use	B1b5 – B2b1	n/a	

Setting a session policy is optional. If not set, there is no maximum number of APDU allowed and the session cannot be refreshed. In short, the default session policy is coded as: '02 0000'

4.3.34.2 Object policy

This section lists all object policies and indicates which policies are applicable for which type of object. Attempting to set policies not allowed for a certain object type leads to failure on object creation.

Table 50. Access rules

Access rule	Description	Position in AR Header	Extension required?	Extension length
RFU	Reserved for future use	B1b8	n/a	
RFU	Reserved for future use	B1b7	n/a	
POLICY_OBJ_FORBID_ALL	Explicitly forbid all operations	B1b6	N	
POLICY_OBJ_ALLOW_SIGN	Allow signature or MAC generation	B1b5	N	

Table 50. Access rules*...continued*

Access rule	Description	Position in AR Header	Extension required?	Extension length
POLICY_OBJ_ALLOW_VERIFY	Allow signature or MAC verification	B1b4	N	
POLICY_OBJ_ALLOW_KA	Allow key agreement	B1b3	N	
POLICY_OBJ_ALLOW_ENC	Allow encryption	B1b2	N	
POLICY_OBJ_ALLOW_DEC	Allow decryption	B1b1	N	
POLICY_OBJ_ALLOW_KDF	Allow key derivation	B2b8	N	
POLICY_OBJ_ALLOW_WRAP	Allow key wrapping (master key)	B2b7	N	
POLICY_OBJ_ALLOW_READ	Allow to read the object	B2b6	N	
POLICY_OBJ_ALLOW_WRITE	Allow to write the object	B2b5	N	
POLICY_OBJ_ALLOW_GEN	Allow to (re)generate the object (only internally)	B2b4	N	
POLICY_OBJ_ALLOW_DELETE	Allow to delete the object	B2b3	N	
POLICY_OBJ_REQUIRE_SM	Require SCP03 or ECKey session secure messaging where secure messaging requires C_MAC and C_ DECRYPTION set.	B2b2	N	
POLICY_OBJ_REQUIRE_PCR_ VALUE	Indicates that access to the object is allowed only if the given PCR object contains a certain value	B2b1	Y	4 bytes PCR object ID 32 bytes PCR value
POLICY_OBJ_ALLOW_ ATTESTATION	Indicates that this object may be used to create attestation statements (i.e. perform attestation of other objects)	B3b8	N	
POLICY_OBJ_ALLOW_ DESFIRE_AUTHENTICATION	Indicates that this object may be used to perform DESFire authentication	B3b7	N	
POLICY_OBJ_ALLOW_ DESFIRE_DUMP_SESSION_ KEYS	Indicates that the DESFire session keys may be dumped to host	B3b6	N	
POLICY_OBJ_ALLOW_ IMPORT_EXPORT	Indicates that this object can be imported or exported	B3b5	N	
RFU	Other values reserved for future use	B3b4 – B4b1	n/a	

4.4 Applet selection

The applet can be selected by sending a GP SELECT command. This command interacts with the JCOP Card Manager and will result in the selection of the SE050 IoT applet.

Table 51. AppletSelect C-APDU

Field	Value	Description
CLA	0x00	
INS	0xA4	
P1	0x04	
P2	0x00	
Lc	0x10	
Payload	0xA000000396545300000001030000000	Applet AID
Le	0x00	

Table 52. AppletSelect R-APDU Body

Value	Description
Applet version	7-byte VersionInfo

Table 53. AppletSelect R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.5 Session management

See Sessions for general information on sessions.

4.5.1 Generic session commands

4.5.1.1 CreateSession

Creates a session on SE050.

Depending on the authentication object being referenced, a specific method of authentication applies. The response needs to adhere to this authentication method.

Table 54. CreateSession C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See Instruction
P1	P1_DEFAULT	See P1
P2	P2_SESSION_CREATE	See P2
Lc	#(Payload)	Payload length.
Payload	TLV[TAG_1]	4-byte authentication object identifier.
Le	0x0C	Expecting TLV with 8-byte session ID.

Table 55. CreateSession R-APDU Body

Value	Description
TLV[TAG_1]	8-byte session identifier.

Table 56. CreateSession R-APDU Trailer

SW	Description
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.

4.5.1.2 ExchangeSessionData

Sets session policies for the current session.

Table 57. ExchangeSessionData C-APDU

Field	Value	Description
CLA	0x80 or 0x84	-
INS	INS_MGMT	See Instruction
P1	P1_DEFAULT	See P1
P2	P2_SESSION_POLICY	See P2
Lc	#(Payload)	Payload length.
Payload	TLV[TAG_1]	Session policies
	C-MAC	If applicable
Le	0x00	-

Table 58. ExchangeSessionData R-APDU Body

Value	Description
R-MAC	Optional, depending on established security level

Table 59. ExchangeSessionData R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.
SW_CONDITIONS_NOT_SATISFIED	Invalid policies

4.5.1.3 ProcessSessionCmd

Requests a command to be processed within a specific session. Note that the applet does not check the validity of the CLA byte of the TLV[TAG_1] payload.

Table 60. ProcessSessionCmd C-APDU

Field	Value	Description
CLA	0x80 or 0x84	-
INS	INS_PROCESS	See Instruction
P1	P1_DEFAULT	See P1
P2	P2_DEFAULT	See P2
Lc	#(Payload)	Payload length.
Payload	TLV[TAG_SESSION_ID]	Session ID
	TLV[TAG_1]	Actual APDU command to be processed. The full command is to be added, including APDU Header and Payload.
Le	0x00	

Table 61. ProcessSessionCmd R-APDU Body

Value	Description
variable	as defined in the specific command section

Table 62. ProcessSessionCmd R-APDU Trailer

SW	Description
variable	as defined in the specific command section

4.5.1.4 RefreshSession

Refreshes a session on SE050, the policy of the running session can be updated; the rest of the session state remains.

Table 63. RefreshSession C-APDU

Field	Value	Description
CLA	0x80	-
INS	INS_MGMT	See Instruction
P1	P1_DEFAULT	See P1
P2	P2_SESSION_REFRESH	See P2
Lc	#(Payload)	Payload length.
	TLV[TAG_POLICY]	Byte array containing the policy to attach to the session. *[Optional]*
Le	-	

Table 64. RefreshSession R-APDU Body

Value	Description
-	

Table 65. RefreshSession R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.5.1.5 CloseSession

Closes a running session.

When a session is closed, it cannot be reopened.

All session parameters are transient.

If CloseSession returns a Status Word different from SW_NO_ERROR, the applet immediately needs to be reselected as further APDUs would not be handled successfully.

Table 66. CloseSession

Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See Instruction
P1	P1_DEFAULT	See P1
P2	P2_SESSION_CLOSE	See P2

Table 67. CloseSession R-APDU Body

Value	Description
None	

Table 68. CloseSession R-APDU Trailer

SW	Description
SW_NO_ERROR	The session is closed successfully.
SW_CONDITIONS_NOT_SATISFIED	The session is not closed successfully.

4.5.2 UserID session operations

4.5.2.1 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).

Table 69. VerifySessionUserID C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See Instruction
P1	P1_DEFAULT	See P1

Table 69. VerifySessionUserID C-APDU...*continued*

Field	Value	Description
P2	P2_SESSION_USERID	See P2
Lc	#(Payload)	Payload length.
	TLV[TAG_1]	UserID value
Le	-	

Table 70. VerifySessionUserID R-APDU Body

Value	Description
-	

Table 71. VerifySessionUserID R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.
SW_CONDITIONS_NOT_SATISFIED	Wrong userID value.

4.5.3 AESKey session operations

4.5.3.1 SCPInitializeUpdate

[SCP03] Section 7.1.1 shall be applied.

The user shall always set the P1 parameter to '00' (KVN = '00').

4.5.3.2 SCPExternalAuthenticate

[SCP03] Section 7.1.2 shall be applied.

4.5.4 ECKey session operations

4.5.4.1 ECKeySessionInternalAuthenticate

Initiates an authentication based on an ECKey Authentication Object. See Section 3.6.3.3 for more information.

The user shall always use key version number = '00' and key identifier = '00'.

Table 72. ECKeySessionInternalAuthenticate C-APDU

Field	Value	Description
CLA	0x84	
INS	0x88	
P1	P1_DEFAULT	Key version number
P2	P2_DEFAULT	Key identifier

Table 72. ECKeySessionInternalAuthenticate C-APDU...*continued*

Field	Value	Description
Lc	#(Payload)	
Payload	TLV[TAG_1]	Input data (see Table 73)
Le	0x00	

Table 73. ECKeySessionInternalAuthenticate C-APDU payload

TAG	SubTag	Length	Value
0xA6		Var	Control Reference Template
	0x4F	5-16	Applet Instance AID
	0x90	3	SCP identifier and parameters
	0x80	1	Key type
	0x81	1	Key length; only 16 bytes are supported (AES128)
0x7F49			
	0xB0	Var	Host key pair public key.
	0xF0	Var	1-byte ECCurve identifier.
0x5F37		Var	ASN.1 signature generated using the host key pair's private key.

Table 74. ECKeySessionInternalAuthenticate R-APDU Body

Value	Description
0x85	16-byte secure element challenge
0x86	16-byte receipt

Table 75. ECKeySessionInternalAuthenticate R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.5.4.2 ECKeySessionGetECKAPublicKey

Gets the public key of the static device key pair (either RESERVED_ID_ECKEY_SESSION or RESERVED_ID_EXTERNAL_IMPORT).

The key identifier used in subTag 0x83 must be either:

- 0x00 for user authentication.
- 0x02 for ImportExternalObject (i.e., single side import) only.

Note that any key identifier value different from 0x02 or 0x00 is RFU, but if passed, it is treated as user authentication (so equal to 0x00).

Table 76. ECKeySessionGetECKAPublicKey C-APDU

Field	Value	Description
CLA	0x84	

Table 76. ECKeySessionGetECKAPublicKey C-APDU...*continued*

Field	Value	Description
INS	0xCA	
P1	0xBF	
P2	0x21	
Lc	#(Payload)	
Payload	TLV[TAG_1]	Input data (see Table 73)
Le	0x00	

Table 77. ECKeySessionGetECKAPublicKey C-APDU payload

TAG	SubTag	Length	Value
0xA6		Var	Control Reference Template
	0x83	2	Key identifier (byte 1) & Key version number (byte 2)

Table 78. ECKeySessionGetECKAPublicKey R-APDU Body

Value	Description
0x7F49	Byte array containing the requested public key (= SE050 Key Agreement public key, either RESERVED_ID_ECKEY_SESSION or RESERVED_ID_EXTERNAL_IMPORT).
0x5F37	Byte array containing the signature over the other (previous) tags.

Table 79. ECKeySessionGetECKAPublicKey R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.6 Module management

4.6.1 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).

Table 80. Lock behavior

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.

This command can only be used in a session that used the credential with identifier RESERVED_ID_TRANSPORT as authentication object.

Table 81. SetLockState C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See Instruction
P1	P1_DEFAULT	See P1
P2	P2_TRANSPORT	See P2
Lc	#(Payload)	
Payload	TLV[TAG_1]	1-byte LockIndicator
	TLV[TAG_2]	1-byte LockState
Le		

Table 82. SetLockState R-APDU Body

Value	Description
None	

Table 83. SetLockState R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.6.2 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_SCP as authentication object.

Note that the default state is SCP_NOT_REQUIRED.

Table 84. SetPlatformSCPRequest C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See Instruction
P1	P1_DEFAULT	See P1
P2	P2_SCP	See P2
Lc	#(Payload)	
Payload	TLV[TAG_1]	1-byte PlatformSCPRequest
Le		

Table 85. SetPlatformSCPRequest R-APDU Body

Value	Description
None	

Table 86. SetPlatformSCPRequest R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.6.3 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.

Table 87. SetAppletFeatures C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See Instruction
P1	P1_DEFAULT	See P1
P2	P2_VARIANT	See P2
Lc	#(Payload)	Payload length
Payload	TLV[TAG_1]	2-byte Variant from AppletConfig

Table 88. SetAppletFeatures R-APDU Body

Value	Description
None	

Table 89. SetAppletFeatures R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.7 Secure Object management

4.7.1 WriteSecureObject

Creates or writes to a Secure Object to the SE050.

Table 90. WriteSecureObject C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_WRITE	See Instruction, possibly containing INS_TRANSIENT and INS_AUTH_OBJ in addition to INS_WRITE.
P1		See P1
P2		See P2
Lc	#(Payload)	Payload Length.
Payload		See Table 93

Table 91. WriteSecureObject R-APDU Body

Value	Description
-	

Table 92. WriteSecureObject R-APDU Trailer

SW	Description
SW_NO_ERROR	The file is created or updated successfully.

Table 93. WriteSecureObject variants

APDU	Reference	Description
WriteECKey	Write ECKey	Write an EC key pair, private key or public key.
WriteRSAKey	Write RSAKey	Write a raw RSA key pair, private key or public key.
WriteSymmKey	Write SYMMKey	Write an AES, DES or HMAC key.
WriteBinary	Write Binary	Write to a binary file.
WriteUserID	Write UserID	Write a userID value.
WriteCounter	Write Counter	Write or increment a monotonic counter.
WritePCR	WriteCPR	Write a PCR value.

Table 93. WriteSecureObject variants...*continued*

APDU	Reference	Description
ImportObject	Import Object	Import an encrypted serialized Secure Object (previously exported)
ImportExternalObject	Import External Object	Import an encrypted serialized Secure Object (externally created)

4.7.1.1 WriteECKey

Write or update an EC key object.

P1KeyType indicates the key type to be created (if the object does not yet exist).

If P1KeyType = 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 SE050.

If the object already exists, P1KeyType is ignored.

If the curve -indicated in TLV[TAG_2]- is not fully instantiated, the writeECKey command will fail.

Note: *For keys using curve ID equal to ID_ECC_ED_25519 or ID_ECC_MONT_DH_25519, check the description about endianness in* Edwards curve byte order*.*

Table 94. WriteECKey C-APDU

Field	Value	Description
P1	P1KeyType | P1_EC	See P1, 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 ECKey) *[Conditional: only when the private key is externally generated and P1KeyType is either P1_KEY_PAIR or P1_PRIVATE]*

Table 94. WriteECKey C-APDU...*continued*

Field	Value	Description
	TLV[TAG_4]	Public key value (see ECKey) *[Conditional: only when the public key is externally generated and P1KeyType is either P1_KEY_PAIR or P1_PUBLIC]*

4.7.1.2 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.

Warning: writing transient RSAkey Secure Objects in CRT mode causes NVM write accesses.

Table 95. WriteRSAKey C-APDU

Field	Value	Description
P1	P1KeyType | P1_RSA	See P1
P2	P2_DEFAULT or P2_RAW	See P2; P2_RAW only in case P1KeyType = P1_KEY_PAIR and TLV[TAG_3] until TLV[TAG_10] is empty and the SE050 must generate a raw RSA key pair; all other cases: P2_DEFAULT.
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 (RSABitLength) *[Conditional: only when the object identifier is not in use yet]*

Table 95. WriteRSAKey C-APDU...*continued*

Field	Value	Description
	TLV[TAG_3]	P component *[Conditional: only when the object identifier is in CRT mode and the key is generated externally and P1KeyType 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 P1KeyType 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 P1KeyType 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 P1KeyType 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 P1KeyType 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_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 it 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.

4.7.1.3 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 requires 8-byte aligned input, so this can only be used when the key has an 8-byte aligned length.

Table 96. WriteSymmKey C-APDU

Field	Value	Description
P1	See above	See P1
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]	4-byte KEK identifier *[Conditional: only when the key value is RFC3394 wrapped]*
	TLV[TAG_3]	Key value, either plain or RFC3394 wrapped.

4.7.1.4 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.

Note: the policy will be applied immediately after the first WriteBinary APDU command. This means that for large Binary files -which require multiple WriteBinary APDUs due to limitation of the APDU buffer size- the subsequent WriteBinary commands need to fulfill the policy that is set in the first WriteBinary command.

Table 97. WriteBinary C-APDU

Field	Value	Description
P1	P1_BINARY	See P1
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_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]*

4.7.1.5 WriteUserID

Creates a UserID object, setting the user identifier value. The policy defines the maximum number of attempts that can be performed as comparison.

Table 98. WriteUserID C-APDU

Field	Value	Description
P1	P1_USERID	See P1
P2	P2_DEFAULT	See P2
	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 AuthenticationObjectPolicies]*
	TLV[TAG_1]	4-byte object identifier.
	TLV[TAG_2]	Byte array containing 4 to 16 bytes user identifier value.

4.7.1.6 WriteCounter

Creates or writes to a counter object.

Counters can only be incremented, not decremented.

When a counter reaches its maximum value (e.g., 0xFFFFFFFF for a 4-byte counter), it 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.

Table 99. WriteCounter C-APDU

Field	Value	Description
P1	P1_COUNTER	See P1
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_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]*

Table 99. WriteCounter C-APDU...*continued*

Field	Value	Description
	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.]*

4.7.1.7 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 are 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.

Table 100. WritePCR C-APDU

Field	Value	Description
P1	P1_PCR	See P1
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_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]*

4.7.1.8 ImportObject

Writes a serialized Secure Object to the SE050 (i.e., “import”). See SecureObjectImportExport for details on the import/export mechanism.

Table 101. ImportObject C-APDU

Field	Value	Description
P1	P1_DEFAULT	See P1
P2	P2_IMPORT	See P2
Payload	TLV[TAG_1]	4-byte identifier.

Table 101. ImportObject C-APDU...*continued*

Field	Value	Description
	TLV[TAG_2]	1-byte RSAKeyComponent *[Conditional: only when the identifier refers to an RSAKey object]*
	TLV[TAG_3]	Serialized object (encrypted).

4.7.2 ImportExternalObject

Note: *The APDU "ImportExternalObject" must not be used without first contacting NXP to avoid potential problems. If you have used or plan to use the APDU "ImportExternalObject," please make sure you contact your NXP representative.*

Combined with the INS_IMPORT_EXTERNAL mask, enables users to send a WriteSecureObject APDU (WriteECKey until WritePCR) protected by the same security mechanisms as an ECKey session. See Secure Object external import for details on the flow of the external import mechanism.

Table 102. ImportExternalObject C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_IMPORT_EXTERNAL	See Instruction
P1	P1_DEFAULT	See P1
P2	P2_DEFAULT	See P2
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
Le	0x08	8 byte Response MAC

The authentication data field includes the same data as defined for the ECKey session Internal Authenticate command; i.e., the host public key and corresponding signature.

The host public key Identifier is the 4-byte identifier of the public part of the key pair used to sign the ephemeral key.

TAG_1 contains a full WriteSecureObject command, including header and payload. This command is wrapped by the session keys derived from the authentication data present in the previous tags. For example, to import an AES Key, the command defined in WriteSymmKey would be passed.

In summary, the ImportExternalObject can be fully pre-computed offcard. The steps to pre-compute a command are the following:

1. Generate the payload for an INTERNAL AUTHENTICATE command as defined by ECKeySessionInternalAuthenticate. This payload is added to tag TAG_IMPORT_AUTH_DATA as is.
2. Add to tag TAG_IMPORT_AUTH_ID the identifier of the host Key Agreement public key.
3. Perform ECDH using the stored private key and the host Key Agreement public key.

4. Assuming a DR.SE equals to 16 bytes of zeroes, derive the master key and the corresponding session keys defined in ECKeySession.
5. Prepare the complete WriteSecureObject command
6. Using the session keys from step 4, wrap the WriteSecureObject command with C-DEC + C-MAC, as defined in ECKey session
7. Add to tag TAG_1 the complete wrapped APDU from the previous step

Note: each ImportExternalObject command executes in its own implicit one-shot session. This means that for each command, all counters and MAC chaining values are assumed to be the initial values as defined in ECKey session.

Table 103. ImportExternalObject R-APDU Body

Value	Description
CMAC	8-byte CMAC over the MAC chaining value + the status word.

Table 104. ImportExternalObject R-APDU Trailer

SW	Description
SW_NO_ERROR	The importExternalObject has finished succesfully.

4.7.3 ReadSecureObject

4.7.3.1 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 an error.
- 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.

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).

Note: for keys using curve ID = ID_ECC_ED_25519 or ID_ECC_MONT_DH_25519, check the explanation in chapter 7.

Table 105. ReadObject C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_READ	See Instruction, in addition to INS_READ, users can set the INS_ATTEST flag. In that case, attestation applies.
P1	P1_DEFAULT	See P1
P2	P2_DEFAULT	See P2
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 BinaryFile object]*
	TLV[TAG_3]	2-byte length *[Optional: default 0]* *[Conditional: only when the object is a BinaryFile object]*
	TLV[TAG_4]	1-byte RSAKeyComponent: 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 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	

Table 106. ReadObject 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 ObjectAttributes).
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].

Table 107. ReadObject R-APDU Trailer

SW	Description
SW_NO_ERROR	The value is read successfully.
SW_CONDITIONS_ NOT_SATISFIED	The value cannot be read.

4.7.3.2 ExportObject

Reads a transient Secure Object from SE050. See SecureObjectImportExport for details on the import/export mechanism.

Table 108. ExportObject C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_READ	See Instruction.
P1	P1_DEFAULT	See P1
P2	P2_EXPORT	See P2
Lc	#(Payload)	Payload Length.
	TLV[TAG_1]	4-byte object identifier
	TLV[TAG_2]	1-byte RSAKeyComponent (only applies to Secure Objects of type RSAKey).
Le	0x00	

Table 109. ExportObject R-APDU Body

Value	Description
TLV[TAG_1]	Byte array containing exported Secure Object data.

Table 110. ExportObject R-APDU Trailer

SW	Description
SW_NO_ERROR	The file is created or updated successfully.

4.7.4 ManageSecureObject

4.7.4.1 ReadType

Get the type of a Secure Object.

Table 111. ReadType C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_READ	See Instruction
P1	P1_DEFAULT	See P1

Table 111. ReadType C-APDU...*continued*

Field	Value	Description
P2	P2_TYPE	See P2
Lc	#(Payload)	
	TLV[TAG_1]	4-byte object identifier.
Le	0x00	

Table 112. ReadType R-APDU Body

Value	Description
TLV[TAG_1]	Type of the Secure Object: one of SecureObjectType
TLV[TAG_2]	TransientIndicator

Table 113. ReadType R-APDU Trailer

SW	Description
SW_NO_ERROR	Data is returned successfully.

4.7.4.2 ReadSize

Get the size of a Secure Object (in bytes):

- For EC keys: the size of the curve is returned, see ECKey for the exact size per curve. It is not possible to read the size of ED25519 (size = 32bytes).
- 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.

Table 114. ReadSize C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_READ	See Instruction
P1	P1_DEFAULT	See P1
P2	P2_SIZE	See P2
Lc	#(Payload)	
	TLV[TAG_1]	4-byte object identifier.
Le	0x00	

Table 115. ReadSize R-APDU Body

Value	Description
TLV[TAG_1]	Byte array containing size.

Table 116. ReadSize R-APDU Trailer

SW	Description
SW_NO_ERROR	Data are returned successfully.
SW_CONDITIONS_NOT_SATISFIED	Data are not returned.

4.7.4.3 ReadIDList

Get a list of present Secure Object identifiers.

The offset in TAG_1 is an 0-based offset in the list of object. As the user does not know how many objects would be returned, the offset needs to be based on the return values from the previous ReadIDList. If the applet only returns a part of the result, it will indicate that more identifiers are available (by setting TLV[TAG_1] in the response to 0x01). The user can then retrieve the next chunk of identifiers by calling ReadIDList with an offset that equals the amount of identifiers listed in the previous response.

Example 1: first ReadIDList command TAG_1=0, response TAG_1=0, TAG_2=complete list

Example 2: first ReadIDList command TAG_1=0, response TAG_1=1, TAG_2=first chunk (m entries) second ReadIDList command TAG_1=m, response TAG_1=1, TAG_2=second chunk (n entries) thirst ReadIDList command TAG_1=(m+n), response TAG_1=0, TAG_2=third last chunk

Table 117. ReadIDList C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_READ	See Instruction
P1	P1_DEFAULT	See P1
P2	P2_LIST	See P2
Lc	#(Payload)	
	TLV[TAG_1]	2-byte offset
	TLV[TAG_2]	1-byte type filter: 1 byte from SecureObjectType or 0xFF for all types.
Le	0x00	

Table 118. ReadIDList R-APDU Body

Value	Description
TLV[TAG_1]	1-byte MoreIndicator

Table 118. ReadIDList R-APDU Body...*continued*

Value	Description
TLV[TAG_2]	Byte array containing 4-byte identifiers.

Table 119. ReadIDList R-APDU Trailer

SW	Description
SW_NO_ERROR	Data is returned successfully.

4.7.4.4 CheckObjectExists

Check if a Secure Object with a certain identifier exists or not.

Table 120. CheckObjectExists C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See Instruction
P1	P1_DEFAULT	See P1
P2	P2_EXIST	See P2
Lc	#(Payload)	
	TLV[TAG_1]	4-byte existing Secure Object identifier.
Le	0x00	

Table 121. CheckObjectExists R-APDU Body

Value	Description
TLV[TAG_1]	1-byte Result

Table 122. CheckObjectExists R-APDU Trailer

SW	Description
SW_NO_ERROR	Data is returned successfully.

4.7.4.5 DeleteSecureObject

Triggers the deletion of a Secure Object. Garbage collection is triggered, the memory will be freed on the next incoming APDU.

If the object origin = ORIGIN_PROVISIONED, an error will be returned and the object is not deleted.

Table 123. DeleteSecureObject C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See Instruction
P1	P1_DEFAULT	See P1

Table 123. DeleteSecureObject C-APDU...*continued*

Field	Value	Description
P2	P2_DELETE_OBJECT	See P2
Lc	#(Payload)	
	TLV[TAG_1]	4-byte existing Secure Object identifier.
Le	-	

Table 124. DeleteSecureObject R-APDU Body

Value	Description
-	

Table 125. DeleteSecureObject R-APDU Trailer

SW	Description
SW_NO_ERROR	The file is created or updated successfully.

4.8 EC curve management

APDUs listed in this section manage operations related to EC curves.

4.8.1 CreateECCurve

Create an EC curve listed in ECCurve.

Table 126. CreateECCurve C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_WRITE	See Instruction
P1	P1_CURVE	See P1
P2	P2_CREATE	See P2
Lc	#(Payload)	
	TLV[TAG_1]	1-byte curve identifier (from ECCurve).
Le		

Table 127. CreateECCurve R-APDU Body

Value	Description
-	

Table 128. CreateECCurve R-APDU Trailer

SW	Description
SW_NO_ERROR	Data is returned successfully.

4.8.2 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.

This function must be called for all supported curves in ECCurve when the curve is to be used, except curve identifiers equal to ID_ECC_ED_25519 and ID_ECC_MONT_DH_25519 (see Note in ECCurve).

Table 129. SetECCurveParam C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_WRITE	See Instruction
P1	P1_CURVE	See P1
P2	P2_PARAM	See P2
Lc	#(Payload)	
	TLV[TAG_1]	1-byte curve identifier, from ECCurve
	TLV[TAG_2]	1-byte ECCurveParam
	TLV[TAG_3]	Bytestring containing curve parameter value.

Table 130. SetECCurveParam R-APDU Body

Value	Description
-	

Table 131. SetECCurveParam R-APDU Trailer

SW	Description
SW_NO_ERROR	Data is returned successfully.

4.8.3 GetECCurveID

Get the curve associated with an EC key.

Table 132. GetECCurveID C-APDU

Field	Value	Description
CLA	0x80	

Table 132. GetECCurveID C-APDU...*continued*

Field	Value	Description
INS	INS_READ	See Instruction
P1	P1_CURVE	See P1
P2	P2_ID	See P2
Lc	#(Payload)	
Payload	TLV[TAG_1]	4-byte identifier
Le	0x00	

Table 133. GetECCurveID R-APDU Body

Value	Description
TLV[TAG_1]	1-byte curve identifier (from ECCurve)

Table 134. GetECCurveID R-APDU Trailer

SW	Description
SW_NO_ERROR	Data is returned successfully.

4.8.4 ReadECCurveList

Get a list of (Weierstrass) EC curves that are instantiated.

Table 135. ReadECCurveList C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_READ	See Instruction
P1	P1_CURVE	See P1
P2	P2_LIST	See P2
Le	0x00	

Table 136. ReadECCurveList R-APDU Body

Value	Description
TLV[TAG_1]	Byte array listing all curve identifiers in ECCurve (excluding UNUSED) where the curve identifier < 0x40; for each curve, a 1-byte SetIndicator is returned.

Table 137. ReadECCurveList R-APDU Trailer

SW	Description
SW_NO_ERROR	Data is returned successfully.

4.8.5 DeleteECCurve

Deletes an EC curve.

Table 138. DeleteECCurve C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See Instruction
P1	P1_CURVE	See P1
P2	P2_DELETE_OBJECT	See P2
Lc	#(Payload)	
	TLV[TAG_1]	1-byte curve identifier (from ECCurve)

Table 139. DeleteECCurve R-APDU Body

Value	Description
-	

Table 140. DeleteECCurve R-APDU Trailer

SW	Description
SW_NO_ERROR	Data is returned successfully.

4.9 Crypto Object management

4.9.1 CreateCryptoObject

Creates a Crypto Object on the SE050. Once the Crypto Object is created, it is bound to the user who created the Crypto Object.

For valid combinations of CryptoObject and the CryptoObject subtype, see CryptoObject.

Table 141. CreateCryptoObject C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_WRITE	See Instruction
P1	P1_CRYPTO_OBJ	See P1
P2	P2_DEFAULT	See P2
Lc	#(Payload)	Payload length
Payload	TLV[TAG_1]	2-byte Crypto Object identifier
	TLV[TAG_2]	1-byte CryptoContext
	TLV[TAG_3]	1-byte Crypto Object subtype, either from DigestMode, CipherMode or MACAlgo (depending on TAG_2).

Table 142. CreateCryptoObject R-APDU Body

Value	Description
-	

Table 143. CreateCryptoObject R-APDU Trailer

SW	Description
SW_NO_ERROR	The file is created or updated successfully.

4.9.2 ReadCryptoObjectList

Get the list of allocated Crypto Objects indicating the identifier, the CryptoContext and the sub type of the CryptoContext.

Table 144. ReadCryptoObjectList C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_READ	See Instruction
P1	P1_CRYPTO_OBJ	See P1
P2	P2_LIST	See P2
Le	0x00	

Table 145. ReadCryptoObjectList R-APDU Body

Value	Description
TLV[TAG_1]	Byte array containing a list of 2-byte Crypto Object identifiers, followed by 1-byte CryptoContext and 1-byte subtype for each Crypto Object (so 4 bytes for each Crypto Object).

Table 146. ReadCryptoObjectList R-APDU Trailer

SW	Description
SW_NO_ERROR	Data is returned successfully.

4.9.3 DeleteCryptoObject

Deletes a Crypto Object on the SE050.

Note: when a Crypto Object is deleted, the memory (as mentioned in Crypto Objects) is de-allocated and will be freed up on the next incoming APDU.

Table 147. DeleteCryptoObject C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See Instruction
P1	P1_CRYPTO_OBJ	See P1

Table 147. DeleteCryptoObject C-APDU...*continued*

Field	Value	Description
P2	P2_DELETE_OBJECT	See P2
Lc	#(Payload)	Payload length
Payload	TLV[TAG_1]	2-byte Crypto Object identifier

Table 148. DeleteCryptoObject R-APDU Body

Value	Description
-	

Table 149. DeleteCryptoObject R-APDU Trailer

SW	Description
SW_NO_ERROR	The file is created or updated successfully.

4.10 Crypto operations EC

Elliptic Curve Crypto operations are supported and tested for all curves listed in ECCurve.

4.10.1 Signature generation

4.10.1.1 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]).

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.

Table 150. ECDSASign C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	Instruction
P1	P1_SIGNATURE	See P1
P2	P2_SIGN	See P2
Lc	#(Payload)	
	TLV[TAG_1]	4-byte identifier of EC key pair or private key.

Table 150. ECDSASign C-APDU...*continued*

Field	Value	Description
	TLV[TAG_2]	1-byte ECSignatureAlgo.
	TLV[TAG_3]	Byte array containing input data.
Le	0x00	Expecting ASN.1 signature

Table 151. ECDSASign R-APDU Body

Value	Description
TLV[TAG_1]	ECDSA Signature in ASN.1 format.

Table 152. ECDSASign R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.10.1.2 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 for TLV[TAG_3] need to be the plain data (i.e. not hashed), maximum length is:

- TBD bytes for use in the default session, an AESKey or an ECKey session.
- TBD bytes for use in a UserID session.

These limits on input data length are not affected by platform SCP.

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).

Note: *See Section 7 for correct byte order.*

Table 153. EdDSASign C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	Instruction
P1	P1_SIGNATURE	See P1
P2	P2_SIGN	See P2
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

Table 154. EdDSASign R-APDU Body

Value	Description
TLV[TAG_1]	EdDSA Signature (r concatenated with s).

Table 155. EdDSASign R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.10.1.3 ECDAASign

The ECDAASign command signs external data using the indicated key pair or private key. This is performed according to ECDAA. The generated signature is:

- r = random mod n
- s = (r + T.d_s) mod n where d is the private key

The ECDAASignatureAlgo indicates the applied algorithm.

This APDU command should be used with a key identifier linked to TPM_ECC_BN_P256 curve.

Note: The applet allows the random input to be 32 bytes of zeroes; the user must take care that this is not considered as valid input. Only input in the interval [1, n-1] must be considered as valid.

Table 156. ECDAASign C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	Instruction
P1	P1_SIGNATURE	See P1
P2	P2_SIGN	See P2
Lc	#(Payload)	
	TLV[TAG_1]	4-byte identifier of EC key pair or private key.
	TLV[TAG_2]	1-byte ECDAASignatureAlgo
	TLV[TAG_3]	*T* = 32-byte array containing hashed input data.
	TLV[TAG_4]	*r* = 32-byte array containing random data, must be in the interval [1, n-1] where n is the order of the curve.
Le	0x00	Expecting signature

Table 157. ECDAASign R-APDU Body

Value	Description
TLV[TAG_1]	ECDAA Signature (r concatenated with s).

Table 158. ECDAASign R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.10.2 Signature verification

4.10.2.1 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.

Table 159. ECDSAVerify C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	Instruction
P1	P1_SIGNATURE	See P1
P2	P2_VERIFY	See P2
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 hashed data to compare.
	TLV[TAG_5]	Byte array containing ASN.1 signature
Le	0x03	Expecting TLV with Result

Table 160. ECDSAVerify R-APDU Body

Value	Description
TLV[TAG_1]	Result of the signature verification (Result).

Table 161. ECDSAVerify R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.
SW_CONDITIONS_NOT_SATISFIED	Incorrect data

4.10.2.2 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.

The input data for TLV[TAG_3] need to be the plain data (i.e. not hashed), maximum length is:

- TBD bytes for use in the default session, an AESKey or an ECKey session.
- TBD bytes for use in a UserID session.

These limits on input data length are not affected by platform SCP.

Note: See chapter Edwards curve byte order for correct byte order as both r and s need to be reversed (converting endianness).

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.

Table 162. EdDSAVerify C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	Instruction
P1	P1_SIGNATURE	See P1
P2	P2_VERIFY	See P2
Lc	#(Payload)	
	TLV[TAG_1]	4-byte identifier of the key pair or public key.
	TLV[TAG_2]	1-byte EDSignatureAlgo.
	TLV[TAG_3]	Byte array containing plain data to compare.
	TLV[TAG_5]	64-byte array containing the signature (concatenation of r and s).
Le	0x03	Expecting TLV with Result

Table 163. EdDSAVerify R-APDU Body

Value	Description
TLV[TAG_1]	Result of the signature verification (Result).

Table 164. EdDSAVerify R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

Table 164. EdDSAVerify R-APDU Trailer...*continued*

SW	Description
SW_CONDITIONS_NOT_SATISFIED	Incorrect data

4.10.3 Shared secret generation

4.10.3.1 ECDHGenerateSharedSecret

The ECDHGenerateSharedSecret command generates a shared secret ECC point on the curve using an EC private key on SE050 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 ID_ECC_ED_25519.

Note that ECDHGenerateSharedSecret commands with EC keys using curve ID_ECC_MONT_DH_25519 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_CONDTIONS_NOT_SATISFIED.

Table 165. ECDHGenerateSharedSecret C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	Instruction
P1	P1_EC	See P1
P2	P2_DH	See P2
Lc	#(Payload)	
Payload	TLV[TAG_1]	4-byte identifier of the key pair or private key.
	TLV[TAG_2]	External public key (see ECKey).
Le	0x00	Expected shared secret length.

Table 166. ECDHGenerateSharedSecret R-APDU Body

Value	Description
TLV[TAG_1]	The returned shared secret.

Table 167. ECDHGenerateSharedSecret R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.11 Crypto operations RSA

RSA crypto operations will be available for certain bit lengths, defined in RSABitLength

For detailed information, see RFC8017 on PKCS#1 RSA Cryptography Specification.

4.11.1 Signature Generation

4.11.1.1 RSASign

The RSASign command signs the input message using an RSA private key.

Padding schemes supported: see RSASignatureAlgo.

When the RSASignatureAlgo is PSS based (ends with '_PSS'), the salt length will be the default length, i.e. equal to the digest length.

Table 168. RSASign C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	Instruction
P1	P1_SIGNATURE	See P1
P2	P2_SIGN	See P2
Lc	#(Payload)	
	TLV[TAG_1]	4-byte identifier of the key pair or private key.
	TLV[TAG_2]	1-byte RSASignatureAlgo
	TLV[TAG_3]	Byte array containing input data.
Le	0x00	Expecting ASN.1 signature.

Table 169. RSASign R-APDU Body

Value	Description
TLV[TAG_1]	RSA signature in ASN.1 format.

Table 170. RSASign R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.11.2 Signature Verification

4.11.2.1 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.

When the RSASignatureAlgo is PSS based (ends with '_PSS'), the salt length will be the default length, i.e. equal to the digest length.

Table 171. RSAVerify C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	Instruction
P1	P1_SIGNATURE	See P1
P2	P2_VERIFY	See P2
Lc	#(Payload)	
Payload		
	TLV[TAG_1]	4-byte identifier of the key pair or public key.
	TLV[TAG_2]	1-byte RSASignatureAlgo
	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

Table 172. RSAVerify R-APDU Body

Value	Description
TLV[TAG_1]	Result: Verification result

Table 173. RSAVerify R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.11.3 Encryption

4.11.3.1 RSAEncrypt

The RSAEncrypt command encrypts data.

Table 174. RSAEncrypt C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	Instruction
P1	P1_RSA	See P1
P2	P2_ENCRYPT_ONESHOT	See P2
Lc	#(Payload)	
Payload	TLV[TAG_1]	4-byte identifier of the key pair or public key.
	TLV[TAG_2]	1-byte RSAEncryptionAlgo
	TLV[TAG_3]	Byte array containing data to be encrypted.
Le	0x00	Expected TLV with encrypted data.

Table 175. RSAEncrypt R-APDU Body

Value	Description
TLV[TAG_1]	Encrypted data

Table 176. RSAEncrypt R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.11.3.2 RSADecrypt

The RSADecrypt command decrypts data.

Table 177. RSADecrypt C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	Instruction
P1	P1_RSA	See P1
P2	P2_DECRYPT_ONESHOT	See P2
Lc	#(Payload)	
Payload	TLV[TAG_1]	4-byte identifier of the key pair or private key.
	TLV[TAG_2]	1-byte RSAEncryptionAlgo
	TLV[TAG_3]	Byte array containing data to be decrypted.
Le	0x00	Expected TLV with decrypted data.

Table 178. RSADecrypt R-APDU Body

Value	Description
TLV[TAG_1]	Decrypted data

Table 179. RSADecrypt R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.12 Crypto operations AES/DES

Cipher operations can be done either using Secure Object of type AESKey or DESKey.

CipherMode indicates the algorithm to be applied.

Cipher operations can be done in one shot mode or in multiple steps. Users are recommended to opt for one shot mode as much as possible as there is no NVM write access in that case, while an AES operation in multiple steps involves NVM write access.

There are 2 options to use AES crypto modes:

- in multiple steps: init/update/final – multiple calls to process data.
- in one shot mode – 1 call to process data

Note: If the Crypto Object is using AES in CTR mode, input data for CipherUpdate need to be block aligned (16-byte blocks).

4.12.1 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.

Table 180. CipherInit C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	Instruction
P1	P1_CIPHER	See P1
P2	P2_ENCRYPT or P2_DECRYPT	See P2
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	-	

Table 181. CipherInit R-APDU Body

Value	Description
-	

Table 182. CipherInit R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.12.2 CipherUpdate

Update a cipher context.

Table 183. CipherUpdate C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	Instruction

Table 183. CipherUpdate C-APDU...*continued*

Field	Value	Description
P1	P1_CIPHER	See P1
P2	P2_UPDATE	See P2
Lc	#(Payload)	
Payload	TLV[TAG_2]	2-byte Crypto Object identifier
	TLV[TAG_3]	Byte array containing input data
Le	0x00	Expecting returned data.

Table 184. CipherUpdate R-APDU Body

Value	Description
TLV[TAG_1]	Output data

Table 185. CipherUpdate R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.12.3 CipherFinal

Finish a sequence of cipher operations.

Table 186. CipherFinal C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	Instruction
P1	P1_CIPHER	See P1
P2	P2_FINAL	See P2
Lc	#(Payload)	
Payload	TLV[TAG_2]	2-byte Crypto Object identifier
	TLV[TAG_3]	Input data
Le	0x00	Expected returned data.

Table 187. CipherFinal R-APDU Body

Value	Description
TLV[TAG_1]	Output data

Table 188. CipherFinal R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.12.4 CipherOneShot

Encrypt or decrypt data in one shot mode.

The key object must be either an AES key or a DES key.

Table 189. CipherOneShot C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	Instruction
P1	P1_CIPHER	See P1
P2	P2_ENCRYPT_ONESHOT or P2_DECRYPT_ONESHOT	See P2
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.

Table 190. CipherOneShot R-APDU Body

Value	Description
TLV[TAG_1]	Output data

Table 191. CipherOneShot R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.13 Message Authentication Codes

There are 2 options to use Message Authentication Codes on SE050:

- in multiple steps: init/update/final – multiple calls to process data.
- in one shot mode – 1 call to process data

Users are recommended to opt for one shot mode as much as possible as there is no NVM write access in that case, while a MAC operation in multiple steps involves NVM write access.

4.13.1 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.

Table 192. MACInit C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	Instruction
P1	P1_MAC	See P1
P2	P2_GENERATE or P2_ VALIDATE	See P2
Lc	#(Payload)	
Payload	TLV[TAG_1]	4-byte identifier of the MAC key.
	TLV[TAG_2]	2-byte Crypto Object identifier
Le	0x00	

Table 193. MACInit R-APDU Body

Value	Description
-	-

Table 194. MACInit R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.13.2 MACUpdate

Update a MAC operation.

Table 195. MACUpdate C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	Instruction
P1	P1_MAC	See P1
P2	P2_UPDATE	See P2
Lc	#(Payload)	
Payload	TLV[TAG_1]	Byte array containing data to be taken as input to MAC.

Table 195. MACUpdate C-APDU...*continued*

Field	Value	Description
	TLV[TAG_2]	2-byte Crypto Object identifier
Le	-	

Table 196. MACUpdate R-APDU Body

Value	Description
-	

Table 197. MACUpdate R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.13.3 MACFinal

Finalize a MAC operation.

Table 198. MACFinal C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	Instruction
P1	P1_MAC	See P1
P2	P2_FINAL	See P2
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 if the crypto object is set for validating (MACInit P2 = P2_VALIDATE)]*
Le	0x00	Expecting MAC or result.

Table 199. MACFinal R-APDU Body

Value	Description
TLV[TAG_1]	MAC value (when MACInit had P2 = P2_GENERATE) or Result (when MACInit had P2 = P2_VERIFY).

Table 200. MACFinal R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.13.4 MACOneShot

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.

Table 201. MACOneShot C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	Instruction
P1	P1_MAC	See P1
P2	P2_GENERATE_ONESHOT or P2_VALIDATE_ONESHOT	See P2
Lc	#(Payload)	
Payload	TLV[TAG_1]	4-byte identifier of the key object.
	TLV[TAG_2]	1-byte MACAlgo
	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.

Table 202. MACOneShot R-APDU Body

Value	Description
TLV[TAG_1]	MAC value (P2=P2_GENERATE_ONESHOT) or Result (when p2=P2_VALIDATE_ONESHOT).

Table 203. MACOneShot R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.14 Key Derivation Functions

4.14.1 HKDF

Perform HMAC Key Derivation Function according to [RFC5869].

The HKDF can only be used in Extract-And-Expand mode. In this mode, the full algorithm is executed. 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. Expand-only mode is not supported.

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.

Table 204. HKDF C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	Instruction
P1	P1_DEFAULT	See P1
P2	P2_HKDF	See P2
Lc	#(Payload)	
	TLV[TAG_1]	4-byte HMACKey identifier (= IKM)
	TLV[TAG_2]	1-byte DigestMode (except DIGEST_NO_HASH and DIGEST_SHA224)
	TLV[TAG_3]	[Optional] Salt. (0 to 64 bytes)
	TLV[TAG_4]	[Optional] Info: The context and information to apply (1 to 448 bytes).
	TLV[TAG_5]	2-byte requested length (L): 1 up to MAX_APDU_PAYLOAD_LENGTH
Le	0x00	

Table 205. HKDF R-APDU Body

Value	Description
TLV[TAG_1]	HKDF output.

Table 206. HKDF R-APDU Trailer

SW	Description
SW_NO_ERROR	The HKDF is executed successfully.

4.14.2 PBKDF2

Password Based Key Derivation Function 2 (PBKDF2) according to [RFC8018] with HMAC SHA1 as underlying pseudorandom function.

The password is an input to the KDF and must be stored inside the SE050.

The output is returned to the host.

4.14.2.1 PBKDF2DeriveKey

Table 207. PBKDF2DeriveKeyC-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	Instruction
P1	P1_DEFAULT	See P1
P2	P2_PBKDF	See P2
Lc	#(Payload)	

Table 207. PBKDF2DeriveKeyC-APDU*...continued*

Field	Value	Description
	TLV[TAG_1]	4-byte password identifier (object type must be HMACKey)
	TLV[TAG_2]	Salt (0 to 64 bytes) *[Optional]*
	TLV[TAG_3]	2-byte Iteration count: 1 up to 0x7FFF.
	TLV[TAG_4]	2-byte Requested length: 1 up to 512 bytes.
Le	0x00	Expecting derived key material.

Table 208. PBKDF2DeriveKey R-APDU Body

Value	Description
TLV[TAG_1]	Derived key material (session key).

Table 209. PBKDF2DeriveKey R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.15 MIFARE DESFire support

MIFARE DESFire EV2 Key derivation (S-mode). This is limited to AES128 keys only.

The SE050 can be used by a card reader to setup a session where the SE050 stores the master key(s) and the session keys are generated and passed to the host.

The SE050 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.

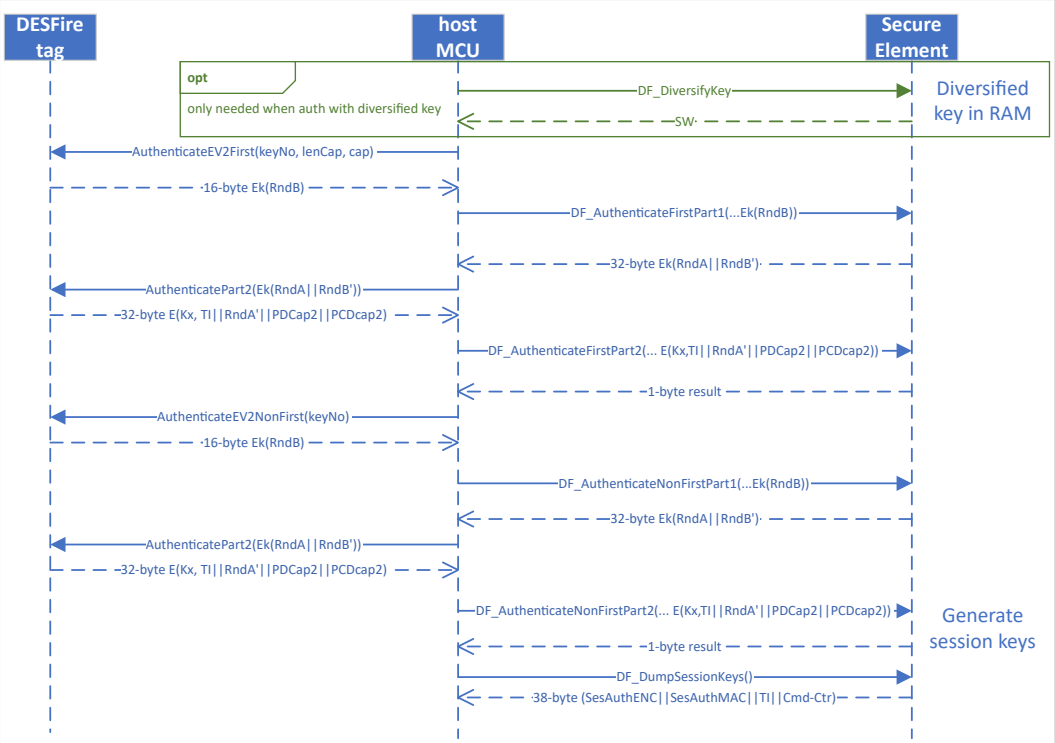

Figure 16. Example DESFire authentication using SE050

The SE050 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..

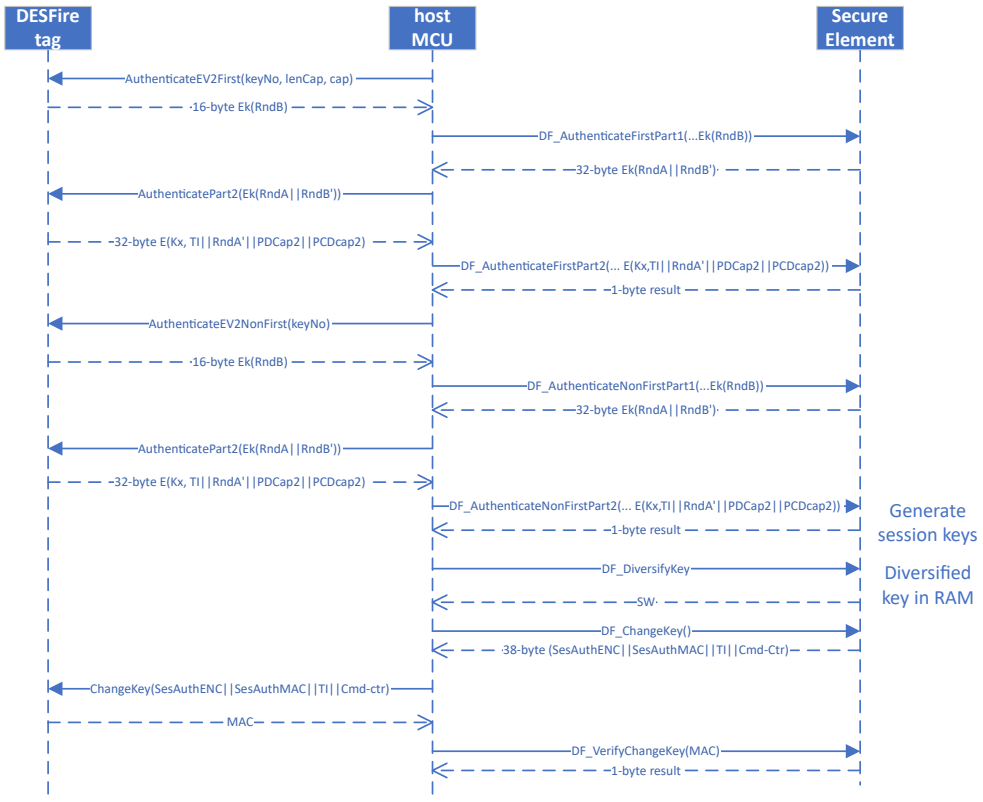

Figure 17. DESFire ChangeKey example

4.15.1 DFDiversifyKey

Create a Diversified Key. Input is *divInput* of 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.

Table 210. DFDiversifyKey C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	Instruction
P1	P1_DEFAULT	See P1
P2	P2_DIVERSIFY	See P2
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		

Table 211. DFDiversifyKey R-APDU Body

Value	Description
-	

Table 212. DFDiversifyKey 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.

4.15.2 DFAuthenticateFirst

Mutual authentication between the reader and the card, part 1.

4.15.2.1 DFAuthenticateFirstPart1

Table 213. DFAuthenticateFirstPart1 C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	Instruction
P1	P1_DEFAULT	See P1
P2	P2_AUTH_FIRST_PART1	See P2
Lc	#(Payload)	
	TLV[TAG_1]	4-byte key identifier.
	TLV[TAG_2]	16-byte encrypted card challenge: E(Kx,RndB)
Le	0x00	

Table 214. DFAuthenticateFirstPart1 R-APDU Body

Value	Description
TLV[TAG_1]	32-byte output data: E(Kx, RandA || RandB')

Table 215. DFAuthenticateFirstPart1 R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.15.2.2 DFAuthenticateFirstPart2

For First part 2, the key identifier is implicitly set to the identifier used for the First authentication. DFAuthenticateFirstPart1 needs to be called before; otherwise an error is returned.

Table 216. DFAuthenticateFirstPart2 C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	Instruction
P1	P1_DEFAULT	See P1
P2	P2_AUTH_FIRST_PART2	See P2
Lc	#(Payload)	
	TLV[TAG_1]	32 byte input: E(Kx,TI||RndA'||PDcap2||PCDcap2)
Le	0x00	

Table 217. DFAuthenticateFirstPart2 R-APDU Body

Value	Description
TLV[TAG_1]	12-byte array returning PDcap2||PCDcap2.

Table 218. DFAuthenticateFirstPart2 R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.15.3 DFAuthenticateNonFirst

Mutual authentication between the reader and the card, part 2.

4.15.3.1 DFAuthenticateNonFirstPart1

Table 219. DFAuthenticateNonFirstPart1 C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	Instruction
P1	P1_DEFAULT	See P1
P2	P2_AUTH_NONFIRST_PART1	See P2
Lc	#(Payload)	
	TLV[TAG_1]	4-byte key identifier.
	TLV[TAG_2]	16-byte encrypted card challenge: E(Kx,RndB)
Le	0x00	

Table 220. DFAuthenticateNonFirstPart1 R-APDU Body

Value	Description
TLV[TAG_1]	32-byte output data: E(Kx, RandA || RandB')

Table 221. DFAuthenticateNonFirstPart1 R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.15.3.2 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.

Table 222. DFAuthenticateNonFirstPart2 C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	Instruction
P1	P1_DEFAULT	See P1
P2	P2_AUTH_NONFIRST_PART2	See P2
Lc	#(Payload)	
	TLV[TAG_1]	16-byte E(Kx, RndA')
Le	0x00	

Table 223. DFAuthenticateNonFirstPart2 R-APDU Body

Value	Description
-	

Table 224. DFAuthenticateNonFirstPart2 R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.
SW_WRONG_DATA	Authentication failed.

4.15.4 DFDumpSessionKeys

Dump the Transaction Identifier and the session keys to the host.

Table 225. DFDumpSessionKeys C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	Instruction
P1	P1_DEFAULT	See P1
P2	P2_DUMP_KEY	See P2
Lc	#(Payload)	

Table 225. DFDumpSessionKeys C-APDU*...continued*

Field	Value	Description
Le	0x2A	Expecting TLV with 38 bytes data.

Table 226. DFDumpSessionKeys R-APDU Body

Value	Description
TLV[TAG_1]	38 bytes: KeyID.SesAuthENCKey || KeyID.SesAuthMACKey || TI || Cmd-Ctr

Table 227. DFDumpSessionKeys R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.15.5 DFChangeKey

4.15.5.1 DFChangeKeyPart1

The DFChangeKeyPart1 command is supporting the function to change keys on the DESFire PICC. The command generates the cryptogram required to perform such operation.

The new key and, if used, the current (or old) key must be stored in the SE050 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 SE050 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.

Table 228. DFChangeKeyPart1 C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	Instruction
P1	P1_DEFAULT	See P1
P2	P2_CHANGE_KEY_PART1	See P2
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.

Table 228. DFChangeKeyPart1 C-APDU...*continued*

Field	Value	Description
	TLV[TAG_5]	1-byte key version
Le	0x00	

Table 229. DFChangeKeyPart1 R-APDU Body

Value	Description
TLV[TAG_1]	Cryptogram holding key data

Table 230. DFChangeKeyPart1 R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.15.5.2 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).

Table 231. DFChangeKeyPart2 C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	See Instruction
P1	P1_DEFAULT	See P1
P2	P2_CHANGE_KEY_PART2	See P2
Lc	#(Payload)	
	TLV[TAG_1]	MAC
Le	0x00	

Table 232. DFChangeKeyPart2 R-APDU Body

Value	Description
TLV[TAG_1]	1-byte Result

Table 233. DFVerifyChangeKeyPart2 R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.15.6 DFKillAuthentication

DFKillAuthentication invalidates any authentication and clears the internal DESFire state. Keys used as input (master keys or diversified keys) are not touched.

Table 234. DFKillAuthentication C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	Instruction
P1	P1_DEFAULT	See P1
P2	P2_KILL_AUTH	See P2
Lc	#(Payload)	

Table 235. DFKillAuthentication R-APDU Body

Value	Description
-	

Table 236. DFKillAuthentication R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.16 TLS handshake support

4.16.1 TLSGenerateRandom

Generates a random that is stored in the SE050 and used by TLSPerformPRF.

Table 237. TLSGenerateRandom C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	See Instruction
P1	P1_TLS	See P1
P2	P2_RANDOM	See P2
Lc	#(Payload)	
Le	0x24	Expecting TLV with 32 bytes data.

Table 238. TLSGenerateRandom R-APDU Body

Value	Description
TLV[TAG_1]	32-byte random value

Table 239. TLSGenerateRandom R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.16.2 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]
- ECDHE_PSK Key Exchange algorithm as defined in [RFC5489]
- EC Key Exchange algorithm as defined in [RFC4492]

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.
- Empty when TLV[TAG_2] is absent or empty.

Table 240. TLSCalculatePreMasterSecret C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	See Instruction
P1	P1_TLS	See P1
P2	P2_PMS	See P2
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	-	

Table 241. TLSCalculatePreMasterSecret R-APDU Body

Value	Description
-	

Table 242. TLSCalculatePreMasterSecret R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.16.3 TLSPerformPRF

The command TLSPerformPRF will compute either:

- the master secret for TLS according to [RFC5246], section 8.1
- key expansion data from a master secret for TLS according to [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 SE050.

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 SE050, generated by TLSGenerateRandom.
- P2_TLS_PRF_SRV_HELLO: pass the serverHelloRandom to calculate a master secret, the clientHelloRandom is in SE050, generated by TLSGenerateRandom.
- P2_TLS_PRF_CLI_RANDOM: pass the clientRandom to generate key expansion data, the serverRandom is in SE050, generated by TLSGenerateRandom.
- P2_TLS_PRF_SRV_RANDOM: pass the serverRandom to generate key expansion data, the clientRandom is in SE050

Table 243. TLSPerformPRF C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	See Instruction
P1	P1_TLS	See P1
P2	See description above.	See P2
Lc	#(Payload)	
	TLV[TAG_1]	4-byte HMACKey identifier.
	TLV[TAG_2]	1-byte DigestMode, except DIGEST_NO_HASH and DIGEST_SHA224
	TLV[TAG_3]	Label (1 to 64 bytes)
	TLV[TAG_4]	32-byte random
	TLV[TAG_5]	2-byte requested length (1 up to 512 bytes)
Le	0x00	

Table 244. TLSPerformPRF R-APDU Body

Value	Description
TLV[TAG_1]	Byte array containing requested output data.

Table 245. TLSPerformPRF R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.17 I2C controller support

The I2C controller support is provided to SE050 users to enable the SE050 as I2C controller. A set of commands can be sent via an APDU to the SE050 after which the SE050 will execute the commands and respond via R-APDU.

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 relative timestamp when the object has been retrieved.

TLV[TAG_3] will hold freshness random data.

TLV[TAG_4] will hold the unique ID of the device.

TLV[TAG_5] will hold the signature over the Values of TLV[TAG_1] up to TLV[TAG_4].

The command set that can be put as part of the TLV[TAG_1] payload of the C-APDU is a byte array consisting out of a concatenation of TLV elements from Table 246.

Only 1 READ command is allowed at the end of the TLV.

Table 246. I2C controller command set TLVs

Instruction	Value	Description
CONFIGURE	0x01	configures the I2C controller; followed by 0x0002 and 2 bytes config. Byte 1: target address Byte 2: clock; 0x00 = 100 kHz, 0x01: 400 kHz
WRITE	0x03	Bytes to be written by the I2C master; followed by 2-byte length indicator + length number of bytes to write.
READ	0x04	Number of bytes to be read by the I2C master; followed by 0x0002 and 2 bytes read length.

- A CONFIGURE command stays valid (i.e., stored in the native library) until the next CONFIGURE is sent, so the configuration of a target is saved.
- The CONFIGURE tag must be the first tag in a command sequence.
- The length of a command sequence is limited to MAX_I2CM_COMMAND_LENGTH. If the command is longer, the applet will return SW_CONDITIONS_NOT_SATISFIED.

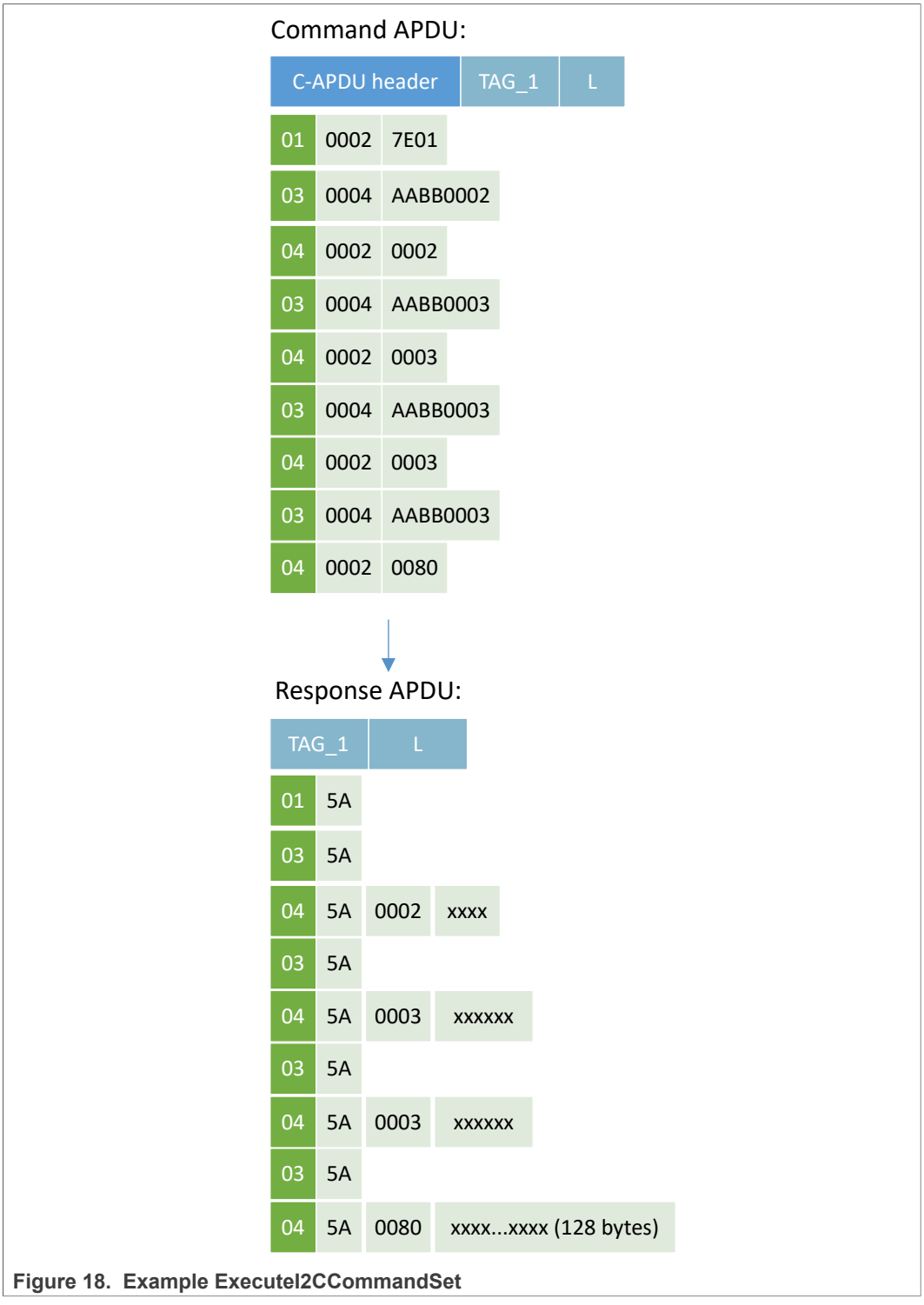

Figure 18. Example ExecuteI2CCommandSet

4.17.1 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 Table 246 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.

Table 247. I2CM_ExecuteCommandSet C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	See Instruction, in addition to INS_CRYPTO, users can set the INS_ATTEST flag. In that case, attestation applies.
P1	P1_DEFAULT	See P1
P2	P2_I2CM	See P2
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 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	Expecting TLV with return data.

Table 248. I2CM_ExecuteCommandSet 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_3]	TLV containing 12-byte timestamp
TLV[TAG_4]	TLV containing 16-byte freshness (random)
TLV[TAG_5]	TLV containing 18-byte chip unique ID
TLV[TAG_6]	TLV containing signature over the concatenated values of TLV[TAG_1], TLV[TAG_3], TLV[TAG_4] and TLV[TAG_5].

Table 249. I2CM_ExecuteCommandSet R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.18 Digest operations

There are 2 options to use Digest operations on SE050:

- in multiple steps: init/update/final – multiple calls to process data.
- in one shot mode – 1 call to process data

Users are recommended to opt for one shot mode as much as possible.

4.18.1 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.

Table 250. DigestInit C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	See Instruction
P1	P1_DEFAULT	See P1
P2	P2_INIT	See P2
Lc	#(Payload)	
	TLV[TAG_2]	2-byte Crypto Object identifier

Table 251. DigestInit R-APDU Body

Value	Description
-	

Table 252. DigestInit R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.18.2 DigestUpdate

Update a digest operation.

Table 253. DigestUpdate C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	See Instruction

Table 253. DigestUpdate C-APDU...*continued*

Field	Value	Description
P1	P1_DEFAULT	See P1
P2	P2_UPDATE	See P2
Lc	#(Payload)	
	TLV[TAG_2]	2-byte Crypto Object identifier
	TLV[TAG_3]	Data to be hashed.
Le		

Table 254. DigestUpdate R-APDU Body

Value	Description
-	-

Table 255. DigestUpdate R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.18.3 DigestFinal

Finalize a digest operation.

Table 256. DigestFinal C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	See Instruction
P1	P1_DEFAULT	See P1
P2	P2_FINAL	See P2
Lc	#(Payload)	
	TLV[TAG_2]	2-byte Crypto Object identifier
	TLV[TAG_3]	Data to be hashed.
Le	0x00	Expecting TLV with hash value.

Table 257. DigestFinal R-APDU Body

Value	Description
TLV[TAG_1]	hash value

Table 258. DigestFinal R-APDU Trailer

SW	Description
SW_NO_ERROR	The hash is created successfully.

4.18.4 DigestOneShot

Performs a hash operation in one shot (without context).

Table 259. DigestOneShot C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	See Instruction
P1	P1_DEFAULT	See P1
P2	P2_ONESHOT	See P2
Lc	#(Payload)	
	TLV[TAG_1]	1-byte DigestMode (except DIGEST_NO_HASH)
	TLV[TAG_2]	Data to hash.
Le	0x00	TLV expecting hash value

Table 260. DigestOneShot R-APDU Body

Value	Description
TLV[TAG_1]	Hash value.

Table 261. DigestOneShot R-APDU Trailer

SW	Description
SW_NO_ERROR	The hash is created successfully.

4.19 Generic management commands

4.19.1 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).

Table 262. GetVersion C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See Instruction
P1	P1_DEFAULT	See P1
P2	P2_VERSION or P2_VERSION_EXT	See P2

Table 262. GetVersion C-APDU...*continued*

Field	Value	Description
Lc	#(Payload)	
Le	0x0B	Expecting TLV with 7-byte data.

Table 263. GetVersion R-APDU Body

Value	Description
TLV[TAG_1]	7-byte VersionInfo

Table 264. GetVersion R-APDU Trailer

SW	Description
SW_NO_ERROR	Data is returned successfully.

4.19.2 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.

Table 265. GetTimestamp C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See Instruction
P1	P1_DEFAULT	See P1
P2	P2_TIME	See P2
Lc	#(Payload)	
Le	0x14	Expecting TLV with timestamp.

Table 266. GetTimestamp R-APDU Body

Value	Description
TLV[TAG_1]	TLV containing a 12-byte operating system timestamp.

Table 267. GetTimestamp R-APDU Trailer

SW	Description
SW_NO_ERROR	Data is returned successfully.

4.19.3 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 since the applet will typically report the “worst case” amount. For example, when allocating 2 bytes at a time, the first report will show 16 bytes being allocated, which remains the same for the next 7 allocations of 2 bytes.

Table 268. GetFreeMemory C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See Instruction
P1	P1_DEFAULT	See P1
P2	P2_MEMORY	See P2
Lc	#(Payload)	
	TLV[TAG_1]	Memory
Le	0x06	Expecting TLV with 2-byte data.

Table 269. GetFreeMemory R-APDU Body

Value	Description
TLV[TAG_1]	2 bytes indicating the amount of free memory of the requested memory type. If 32768 bytes or more bytes are available, 0x7FFF is given as response.

Table 270. GetFreeMemory R-APDU Trailer

SW	Description
SW_NO_ERROR	Data is returned successfully.

4.19.4 GetRandom

Gets random data from the SE050.

Table 271. GetRandom C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See Instruction
P1	P1_DEFAULT	See P1
P2	P2_RANDOM	See P2
Lc	#(Payload)	
	TLV[TAG_1]	2-byte requested size.
Le	0x00	Expecting random data

Table 272. GetRandom R-APDU Body

Value	Description
TLV[TAG_1]	Random data.

Table 273. GetRandom R-APDU Trailer

SW	Description
SW_NO_ERROR	Data is returned successfully.

4.19.5 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).

Table 274. DeleteAll C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See Instruction
P1	P1_DEFAULT	See P1
P2	P2_DELETE_ALL	See P2
Lc	0x00	

Table 275. DeleteAll R-APDU Body

Value	Description
-	

Table 276. DeleteAll R-APDU Trailer

SW	Description
SW_NO_ERROR	Data is returned successfully.

5 APDU list summary

This section contains a list of all C-APDUs.

Table 277. APDU list

Name	CLA	INS	P1	P2	Reference
CreateSession	0x80	0x04	0x00	0x1B	
ExchangeSessionData	0x80	0x04	0x00	0x1F	
ProcessSessionCmd	0x80	0x05	0x00	0x00	
RefreshSession	0x80	0x04	0x00	0x1E	
CloseSession	0x80	0x04	0x00	0x1C	
VerifySessionUserID	0x80	0x04	0x00	0x2C	
SCP_InitializeUpdate	0x80	0x50			
SCP_ExternalAuthenticate	0x80	0x82			
ECKeySessionInternalAuthenticate	0x80	0x88			
ECKeySessionGetECKAPublicKey	0x80	0xCA			
SetLockState	0x80	0x04	0x00	0x3E	
SetAppletFeatures	0x80	0x04	0x00	0x3F	
WriteECKey	0x80	0x01	0x01	0x00	
WriteRSAKey	0x80	0x01	0x02	0x00	
WriteSymmKey	0x80	0x01	Type	0x00	
WriteBinary	0x80	0x01	0x06	0x00	
WriteUserID	0x80	0x01	0x07	0x00	
WriteCounter	0x80	0x01	0x08	0x00	
WritePCR	0x80	0x01	0x09	0x00	
ImportObject	0x80	0x01	0x00	0x18	
ReadObject	0x80	0x02	0x00	0x00	
ExportObject	0x80	0x02	0x00	0x19	
ReadType	0x80	0x02	0x00	0x26	
ReadSize	0x80	0x02	0x00	0x07	
ReadIDList	0x80	0x02	0x00	0x25	
CheckObjectExists	0x80	0x04	0x00	0x27	
DeleteSecureObject	0x80	0x04	0x00	0x28	
CreateECCurve	0x80	0x01	0x0B	0x04	
SetECCurveParam	0x80	0x01	0x0B	0x40	
GetECCurveId	0x80	0x02	0x0B	0x36	
ReadECCurveList	0x80	0x02	0x0B	0x25	
DeleteECCurve	0x80	0x04	0x0B	0x28	
ECDSASign	0x80	0x03	0x0C	0x09	

Table 277. APDU list*...continued*

Name	CLA	INS	P1	P2	Reference
EdDSASign	0x80	0x03	0x0C	0x09	
ECDAASign	0x80	0x03	0x0C	0x09	
ECDSAVerify	0x80	0x03	0x0C	0x0A	
EdDSAVerify	0x80	0x03	0x0C	0x0A	
ECDHGenerateSharedSecret	0x80	0x03	0x01	0x0F	
RSASign	0x80	0x03	0x0C	0x09	
RSAVerify	0x80	0x03	0x0C	0x0A	
RSAEncrypt	0x80	0x03	0x0E	0x42	
RSADecrypt	0x80	0x03	0x0E	0x43	
CipherInit	0x80	0x03	0x0E	0x42/0x43	
CipherUpdate	0x80	0x03	0x0E	0x0C	
CipherFinal	0x80	0x03	0x0E	0x0D	
CipherOneShot	0x80	0x03	0x0E	0x37/0x38	
MACInit	0x80	0x03	0x0D	0x03	
MACUpdate	0x80	0x03	0x0D	0x0C	
MACFinal	0x80	0x03	0x0D	0x0D	
MACOneShot	0x80	0x03	0x0D	0x45/0x46	
HKDF	0x80	0x03	0x00	0x2D	
PBKDF2	0x80	0x03	0x00	0x2E	
DFDiversifyKey	0x80	0x03	0x00	0x10	
DFAuthenticateFirstPart1	0x80	0x03	0x00	0x11	
DFAuthenticateFirstPart2	0x80	0x03	0x00	0x12	
DFAuthenticateNonFirstPart2	0x80	0x03	0x00	0x13	
DFDumpSessionKeys	0x80	0x03	0x00	0x14	
DFChangeKeyPart1	0x80	0x03	0x00	0x15	
DFChangeKeyPart2	0x80	0x03	0x00	0x16	
DFKillAuthentication	0x80	0x03	0x00	0x17	
TLSGenerateRandom	0x80	0x03	0x0F	0x49	
TLSCalculatePreMasterSecret	0x80	0x03	0x0F	0x4A	
TLSPerformPRF	0x80	0x03	0x0F	0x4B-0x4E	
I2CM_ExecuteCommandSet	0x80	0x03	0x00	0x30	
DigestInit	0x80	0x03	0x00	0x0B	
DigestUpdate	0x80	0x03	0x00	0x0C	
DigestFinal	0x80	0x03	0x00	0x0D	
DigestOneShot	0x80	0x03	0x00	0x0E	
GetVersion	0x80	0x04	0x00	0x20	

Table 277. APDU list*...continued*

Name	CLA	INS	P1	P2	Reference
GetTimestamp	0x80	0x04	0x00	0x3D	
GetFreeMemory	0x80	0x04	0x00	0x22	
GetRandom	0x80	0x04	0x00	0x49	
GetCryptoObjectList	0x80	0x04	0x00	0x47	
DeleteAll	0x80	0x04	0x00	0x2A	
DeleteCrypto	0x80	0x04	0x00	0x48	

6 Policy mapping

6.1 Policy mapping tables

6.1.1 Policy mapping to symmetric key Secure Objects

Table 278. Policy mapping symmetric key Authentication Objects

Policy	AESKey	DESKey	HMACKey
POLICY_OBJ_ALLOW_SIGN	-	-	-
POLICY_OBJ_ALLOW_ VERIFY	-	-	-
POLICY_OBJ_ALLOW_KA	-	-	-
POLICY_OBJ_ALLOW_ENC	-	-	-
POLICY_OBJ_ALLOW_DEC	-	-	-
POLICY_OBJ_ALLOW_KDF	-	-	-
POLICY_OBJ_ALLOW_WRAP	-	-	-
POLICY_OBJ_ALLOW_READ	-	-	-
POLICY_OBJ_ALLOW_WRITE	writeSymmKey (update)	writeSymmKey (update)	writeSymmKey (update)
POLICY_OBJ_ALLOW_GEN	writeSymmKey (update)	writeSymmKey (update)	writeSymmKey (update)
POLICY_OBJ_ALLOW_ DELETE	DeleteObject	DeleteObject	DeleteObject
POLICY_OBJ_REQUIRE_SM	Any access to the object requires secure messaging.	Any access to the object requires secure messaging.	Any access to the object requires secure messaging.
POLICY_OBJ_REQUIRE_ PCR_VALUE	Any access to the object requires a PCR value.	Any access to the object requires a PCR value.	Any access to the object requires a PCR value.
POLICY_OBJ_ALLOW_ ATTESTATION	-	-	-
POLICY_OBJ_ALLOW_ DESFIRE_AUTHENTICATION	-	-	-
POLICY_OBJ_ALLOW_ DESFIRE_DUMP_SESSION_ KEY	-	-	-
POLICY_OBJ_ALLOW_ IMPORT_EXPORT	-	-	-

Table 279. Policy mapping symmetric key non-Authentication Objects

Policy	AESKey	DESKey	HMACKey
POLICY_OBJ_ALLOW_SIGN	MACOneShot MACInit	MACOneShot MACInit	MACOneShot MACInit

Table 279. Policy mapping symmetric key non-Authentication Objects*...continued*

Policy	AESKey	DESKey	HMACKey
POLICY_OBJ_ALLOW_ VERIFY	-	-	-
POLICY_OBJ_ALLOW_KA	-	-	-
POLICY_OBJ_ALLOW_ENC	CipherOneShot CipherInit	CipherOneShot CipherInit	-
POLICY_OBJ_ALLOW_DEC	CipherOneShot CipherInit	CipherOneShot CipherInit	-
POLICY_OBJ_ALLOW_KDF	-	-	HKDF PBKDF2 TLSPerformPRF
POLICY_OBJ_ALLOW_WRAP	on Key Encryption Key for writeSymmKey (create/update)	on Key Encryption Key for writeSymmKey (create/update)	on Key Encryption Key for writeSymmKey (create/update)
POLICY_OBJ_ALLOW_READ	-	-	-
POLICY_OBJ_ALLOW_WRITE	writeSymmKey (update)	writeSymmKey (update)	writeSymmKey (update)
POLICY_OBJ_ALLOW_GEN	-	-	-
POLICY_OBJ_ALLOW_ DELETE	DeleteObject	DeleteObject	DeleteObject
POLICY_OBJ_REQUIRE_SM	Any access to the object requires secure messaging.	Any access to the object requires secure messaging.	Any access to the object requires secure messaging.
POLICY_OBJ_REQUIRE_ PCR_VALUE	Any access to the object requires a PCR value.	Any access to the object requires a PCR value.	Any access to the object requires a PCR value.
POLICY_OBJ_ALLOW_ ATTESTATION	-	-	-
POLICY_OBJ_ALLOW_ DESFIRE_AUTHENTICATION	DFDiversifyKey DFAuthenticateFirstPart1 DFAuthenticateNonFirstPart1 DFChangeKeyPart1	-	-
POLICY_OBJ_ALLOW_ DESFIRE_DUMP_SESSION_ KEY	DFDumpSessionKey	-	-
POLICY_OBJ_ALLOW_ IMPORT_EXPORT	ImportObject ExportObject	ImportObject ExportObject	-

6.1.2 Policy mapping to RSAKey Secure Objects

Table 280. Policy mapping

Policy	RSA Keypair	RSA Public key	RSA Private Key
POLICY_OBJ_ALLOW_SIGN	RSASign	-	RSASign
POLICY_OBJ_ALLOW_ VERIFY	RSAVerify	RSAVerify	-
POLICY_OBJ_ALLOW_KA	TLsCalculatePreMasterSecret	-	-
POLICY_OBJ_ALLOW_ENC	RSAEncrypt CipherOneShot CipherInit	RSAEncrypt CipherOneShot CipherInit	-
POLICY_OBJ_ALLOW_DEC	RSADecrypt CipherOneShot CipherInit	-	RSADecrypt CipherOneShot CipherInit
POLICY_OBJ_ALLOW_KDF	-	-	-
POLICY_OBJ_ALLOW_WRAP	-	-	-
POLICY_OBJ_ALLOW_READ	ReadObject (value)	ReadObject (value)	-
POLICY_OBJ_ALLOW_WRITE	writeRSAKey (update)	writeRSAKey (update)	writeRSAKey (update)
POLICY_OBJ_ALLOW_GEN	Regenerate	-	-
POLICY_OBJ_ALLOW_ DELETE	DeleteObject	DeleteObject	DeleteObject
POLICY_OBJ_REQUIRE_SM	Any access to the object requires secure messaging.	Any access to the object requires secure messaging.	Any access to the object requires secure messaging.
POLICY_OBJ_REQUIRE_ PCR_VALUE	Any access to the object requires a PCR value.	Any access to the object requires a PCR value.	Any access to the object requires a PCR value.
POLICY_OBJ_ALLOW_ ATTESTATION	-	-	-
POLICY_OBJ_ALLOW_ DESFIRE_AUTHENTICATION	-	-	-
POLICY_OBJ_ALLOW_ DESFIRE_DUMP_SESSION_ KEY	-	-	-
POLICY_OBJ_ALLOW_ IMPORT_EXPORT	ImportObject ExportObject	ImportObject ExportObject	ImportObject ExportObject

6.1.3 Policy mapping to ECKey Secure Objects

Table 281. Policy mapping ECKey Authentication Objects

Policy	EC Keypair	EC public key	EC private key
POLICY_OBJ_ALLOW_SIGN	-	-	-
POLICY_OBJ_ALLOW_ VERIFY	-	-	-

Table 281. Policy mapping ECKey Authentication Objects*...continued*

Policy	EC Keypair	EC public key	EC private key
POLICY_OBJ_ALLOW_KA	-	-	-
POLICY_OBJ_ALLOW_ENC	-	-	-
POLICY_OBJ_ALLOW_DEC	-	-	-
POLICY_OBJ_ALLOW_KDF	-	-	-
POLICY_OBJ_ALLOW_WRAP	-	-	-
POLICY_OBJ_ALLOW_READ	ReadObject (value)	ReadObject (value)	-
POLICY_OBJ_ALLOW_WRITE	WriteECKey (update)	WriteECKey (update)	WriteECKey (update)
POLICY_OBJ_ALLOW_GEN	Regenerate	-	-
POLICY_OBJ_ALLOW_ DELETE	DeleteObject	DeleteObject	DeleteObject
POLICY_OBJ_REQUIRE_SM	Any access to the object requires SM	Any access to the object requires SM	Any access to the object requires SM
POLICY_OBJ_REQUIRE_ PCR_VALUE	Any access to the object requires a PCR value.	Any access to the object requires a PCR value.	Any access to the object requires a PCR value.
POLICY_OBJ_ALLOW_ ATTESTATION	-	-	-
POLICY_OBJ_ALLOW_ DESFIRE_AUTHENTICATION	-	-	-
POLICY_OBJ_ALLOW_ DESFIRE_DUMP_SESSION_ KEY	-	-	-
POLICY_OBJ_ALLOW_ IMPORT_EXPORT	-	-	-

Table 282. Policy mapping ECKey non-Authentication Objects

Policy	EC Keypair	EC public key	EC private key
POLICY_OBJ_ALLOW_SIGN	ECDSASign EdDSASign ECDAASign	-	ECDSASign EdDSASign ECDAASign
POLICY_OBJ_ALLOW_ VERIFY	ECDSAVerify EdDSAVerify	ECDSAVerify EdDSAVerify	-
POLICY_OBJ_ALLOW_KA	ECDHGenerateSha redSecret TLsCalculatePreMa sterSecret	-	ECDHGenerateSha redSecret
POLICY_OBJ_ALLOW_ENC	-	-	-
POLICY_OBJ_ALLOW_DEC	-	-	-
POLICY_OBJ_ALLOW_KDF	-	-	-
POLICY_OBJ_ALLOW_WRAP	-	-	-

Table 282. Policy mapping ECKey non-Authentication Objects*...continued*

Policy	EC Keypair	EC public key	EC private key
POLICY_OBJ_ALLOW_READ	ReadObject (value)	ReadObject (value)	-
POLICY_OBJ_ALLOW_WRITE	WriteECKey (update)	WriteECKey (update)	WriteECKey (update)
POLICY_OBJ_ALLOW_GEN	Regenerate	-	-
POLICY_OBJ_ALLOW_ DELETE	DeleteObject	DeleteObject	DeleteObject
POLICY_OBJ_REQUIRE_SM	Any access to the object requires SM	Any access to the object requires SM	Any access to the object requires SM
POLICY_OBJ_REQUIRE_ PCR_VALUE	Any access to the object requires a PCR value.	Any access to the object requires a PCR value.	Any access to the object requires a PCR value.
POLICY_OBJ_ALLOW_ ATTESTATION	ReadObject ReadObjectAttributes	-	ReadObject ReadObjectAttributes
POLICY_OBJ_ALLOW_ DESFIRE_AUTHENTICATION	-	-	-
POLICY_OBJ_ALLOW_ DESFIRE_DUMP_SESSION_ KEY	-	-	-
POLICY_OBJ_ALLOW_ IMPORT_EXPORT	ImportObject ExportObject	ImportObject ExportObject	ImportObject ExportObject

6.1.4 Policy mapping to File Secure Objects

Table 283. Policy mapping

Policy	Binary file	UserID	Counter	PCR
POLICY_OBJ_ALLOW_ SIGN	-	-	-	-
POLICY_OBJ_ALLOW_ VERIFY	-	-	-	-
POLICY_OBJ_ALLOW_ KA	-	-	-	-
POLICY_OBJ_ALLOW_ ENC	-	-	-	-
POLICY_OBJ_ALLOW_ DEC	-	-	-	-
POLICY_OBJ_ALLOW_ KDF	-	-	-	-
POLICY_OBJ_ALLOW_ WRAP	-	-	-	-
POLICY_OBJ_ALLOW_ READ	ReadObject (value)	-	ReadObject (value)	ReadObject (value)
POLICY_OBJ_ALLOW_ WRITE	WriteBinary (update)	WriteUserI D	WriteCounter (increment and update)	WritePCR (update)

Table 283. Policy mapping*...continued*

Policy	Binary file	UserID	Counter	PCR
POLICY_OBJ_ALLOW_GEN	-	-	-	-
POLICY_OBJ_ALLOW_DELETE	DeleteObject	DeleteObject	DeleteObject	DeleteObject
POLICY_OBJ_REQUIRE_SM	Any access to the object requires SM	Any access to the object requires SM	Any access to the object requires SM	Any access to the object requires SM
POLICY_OBJ_REQUIRE_PCR_VALUE	Any access to the object requires a PCR value.	Any access to the object requires a PCR value.	Any access to the object requires a PCR value.	Any access to the object requires a PCR value.
POLICY_OBJ_ALLOW_ATTESTATION	-	-	-	-
POLICY_OBJ_ALLOW_DESFIRE_AUTHENTICATION	-	-	-	-
POLICY_OBJ_ALLOW_DESFIRE_DUMP_SESSION_KEY	-	-	-	-
POLICY_OBJ_ALLOW_IMPORT_EXPORT	-	-	-	-

6.2 Non-policy-controlled APDUs

Table 284. Non-policy-controlled APDUs

APDU	Remark
CreateSession	allowed if using an auth object
ExchangeSessionData	
ProcessSessionCmd	
RefreshSession	Covered by session policies
CloseSession	
VerifySessionUserID	Allowed if using an auth object
SCPInitializeUpdate	Allowed if using an auth object
SCPExternalAuthenticate	Allowed if using an auth object
ECKeySessionInternalAuthenticate	Allowed if using an auth object
ECKeySession_GetECKAPublicKey	Allowed if using an auth object
SetAppletFeatures	Allowed if using credential RESERVED_ID_FEATURE
SetPlatformSCPRequest	Allowed if using credential RESERVED_ID_PLATFORM_SCP
SetLockState	Allowed if using credential RESERVED_ID_TRANSPORT

Table 284. Non-policy-controlled APDUs*...continued*

APDU	Remark
ReadType	
ReadSize	
ReadIDList	
CheckObjectExists	
Create/Set/Get/Delete ECCurve	
DFAuthenticateFirstPart2	
DFAuthenticateNonFirstPart2	
DFChangeKeyPart2	
DFKillAuthentication	
TLSGenerateRandom	
I2CM_ExecuteCommandSet	
Digest Init/Update/Final/OneShot	
GetVersion	
GetFreeMemory	
GetTimestamp	
GetCryptoObjectList	
DeleteAll	Allowed if using credential RESERVED_ID_FACTORY_RESET

7 Edwards curve byte order

For keys and key operations using Edwards curve Curve25519, the byte order needs attention as the SE050 uses big endian byte order for most of the parameters on these curves while the standards (RFC8032 and RFC7748) use little endian notation for all parameters.

This applies to WriteECKey (using curve ID_ECC_ED_25519) and will impact:

- EdDSASign/EdDSAVerify (using curve ID_ECC_ED_25519)
- ECDHGenerateSharedSecret (using curve ID_ECC_MONT_DH_25519)

7.1 EdDSA

See Figure 19 for the correct byte order: for the public key and the signature components r and s, the byte order needs to be reversed.

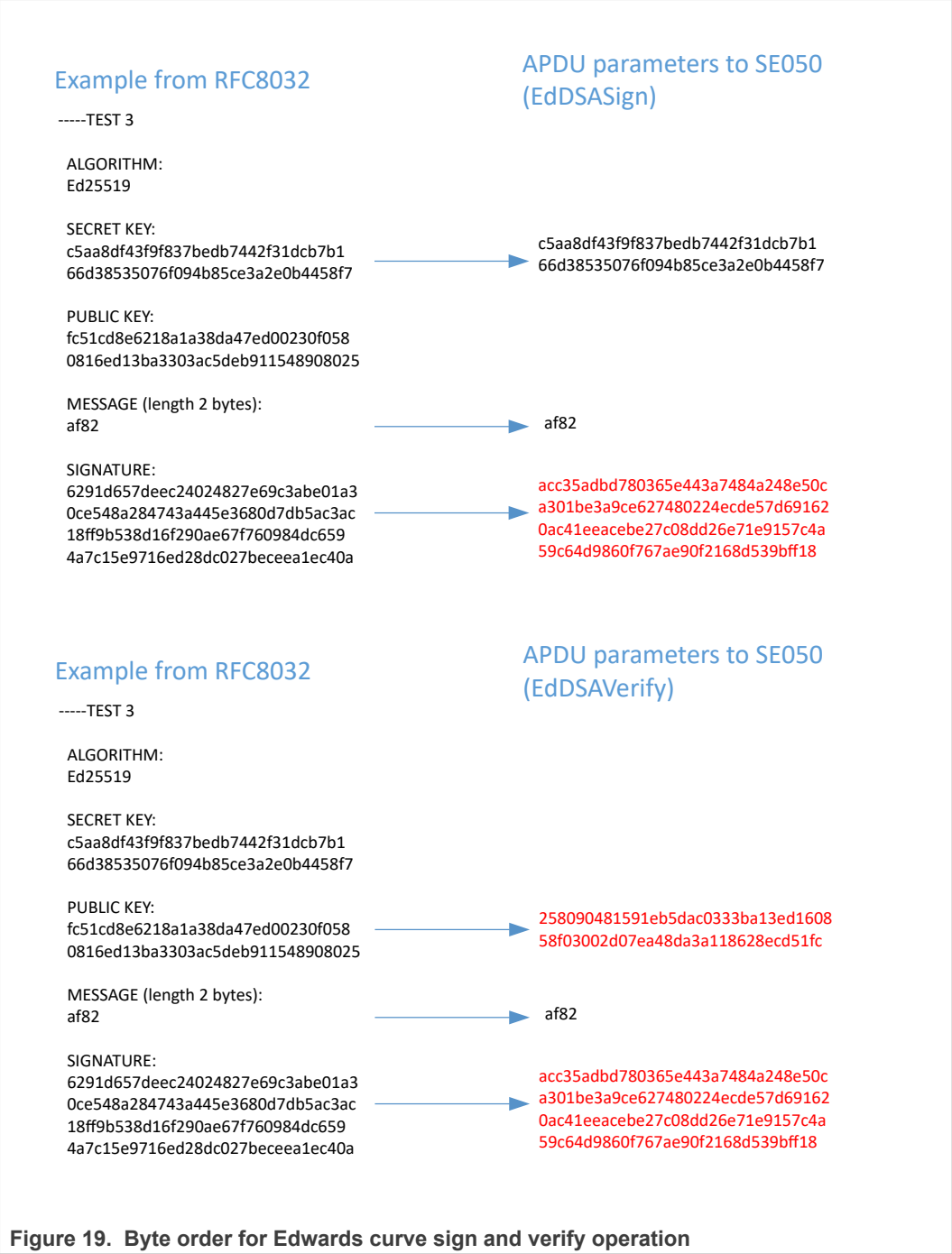

Figure 19. Byte order for Edwards curve sign and verify operation

7.2 ECDHGenerateSharedSecret

See Figure 20 for the correct byte order: for the private key and the shared secret, the byte order needs to be reversed.

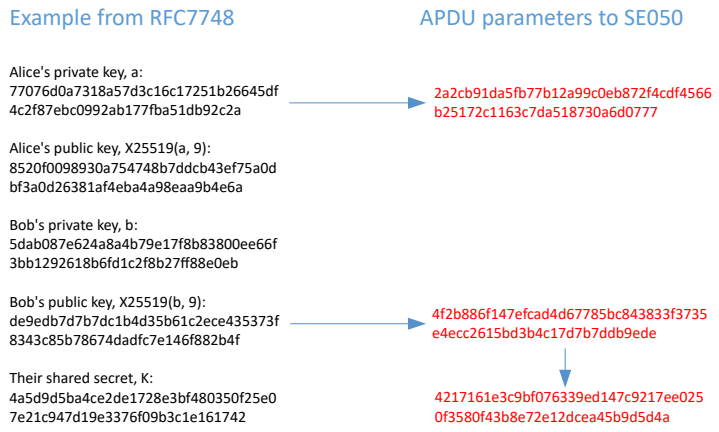

Figure 20. Byte order for Edwards Montgomery curve shared secret operation

8 Memory consumption

8.1 Secure Objects

Note that the values listed in the table are indicative only: they apply to regular Secure Objects (not authentication objects) with a default policy. For EC key objects, the memory for creating the curve needs to be incorporated once (when the curve is created).

Table 285. Secure Object memory for asymmetric keys

Object Type (#bytes NVM/RAM)	Persistent key pair [bytes]	Transient key pair [bytes]	Persistent private key [bytes]	Transient private key [bytes]	Persistent public key [bytes]	Transient public key [bytes]
EC NIST P192 [curve: 252/0]	320/0	208/96	320/0	208/96	204/0	140/64
EC NIST P224 [curve: 276/0]	328/0	208/112	328/0	208/112	212/0	140/80
EC NIST P256 [curve: 260/0]	352/0	208/128	352/0	208/128	220/0	140/80
EC NIST P384 [curve: 396/0]	400/0	208/176	400/0	208/176	252/0	140/112
EC NIST P521 [curve: 504/0]	452/0	208/240	452/0	208/240	288/0	140/144
EC Brainpool160_R1 [curve: 228/0]	312/0	208/96	312/0	208/96	196/0	140/64
EC Brainpool192_R1 [curve: 252/0]	320/0	208/96	320/0	208/96	204/0	140/64
EC Brainpool224_R1 [curve: 276/0]	328/0	208/112	328/0	208/112	212/0	140/80
EC Brainpool256_R1 [curve: 300/0]	352/0	208/128	352/0	208/128	220/0	140/80
EC Brainpool320_R1 [curve: 348/0]	368/0	208/144	368/0	208/144	236/0	140/96
EC Brainpool384_R1 [curve: 396/0]	400/0	208/176	400/0	208/176	252/0	140/112
EC Brainpool512_R1 [curve: 492/0]	448/0	208/224	448/0	208/224	284/0	140/144
EC SEC_P160_K1 [curve: 228/0]	312/0	208/96	312/0	208/96	196/0	140/64
EC SEC_P192_K1 [curve: 252/0]	320/0	208/96	320/0	208/96	204/0	140/64
EC SEC_P224_K1 [curve: 276/0]	328/0	208//112	328/0	208/112	212/0	140/80
EC SEC_P256_K1 [curve: 300/0]	352/0	208/128	352/0	208/128	220/0	140/80
TPM_ECC_BN_P256 [curve: 300/0]	352/0	208/128	352/0	208/128	220/0	140/80

Table 285. Secure Object memory for asymmetric keys *...continued*

Object Type (#bytes NVM/RAM)	Persistent key pair [bytes]	Transient key pair [bytes]	Persistent private key [bytes]	Transient private key [bytes]	Persistent public key [bytes]	Transient public key [bytes]
ED_25519 [curve: 0/0]	308/0	184/112	308/0	184/112	308/0	184/112
MONT_DH_25519 [curve: 0/0]	276/0	184/80	276/0	184/80	276/0	184/96

Table 286. Secure Object memory for asymmetric keys

Object Type (#bytes NVM/RAM)	Persistent key pair [bytes]	Transient key pair [bytes]	Persistent private key [bytes]	Transient private key [bytes]	Persistent public key [bytes]	Transient public key [bytes]
RSA512 raw	412/0	196/240	264/0	132/160	204/0	132/96
RSA512 CRT	536/0	200/368	388/0	136/272	204/0	132/96
RSA1024 raw	604/0	196/432	400/0	140/288	276/0	140/176
RSA1024 CRT	760/0	212/592	556/0	144/448	Not applicable	Not applicable
RSA1152 raw	664/0	208/496	432/0	140/320	292/0	140/192
RSA1152 CRT	788/0	212/608	556/0	144/448	Not applicable	Not applicable
RSA2048 raw	1000/0	208/832	656/0	140/544	404/0	140/304
RSA2048 CRT	1220/0	212/1040	876/0	144/768	Not applicable	Not applicable
RSA3072 raw	1384/0	Not applicable	912/0	Not applicable	532/0	140/432
RSA3072 CRT	1668/0	Not applicable	1196/0	Not applicable	Not applicable	Not applicable
RSA4096 raw	1768/0	Not applicable	1186/0	Not applicable	660/0	Not applicable
RSA4096 CRT	2116/0	Not applicable	1516/0	Not applicable	Not applciable	Not applicable

Table 287. Secure Object memory SymmKey

Object Type	Persistent key [bytes]	Transient key [bytes]
AESKey	NVM: 132 + key size in bytes RAM: 0	NVM: 112 RAM: 16 + key size in bytes
DESKey	NVM: 156 + key size in bytes RAM: 0	NVM: 132 RAM: 32 + key size in bytes

Table 287. Secure Object memory SymmKey...*continued*

Object Type	Persistent key [bytes]	Transient key [bytes]
HMACKey	NVM: 136 + key size in bytes RAM: 0	NVM: 116 RAM: 16 + key size in bytes

Table 288. Secure Object memory File objects

Object Type	Persistent object [bytes]	Transient object [bytes]
BinaryFile	NVM: 92 + file size in bytes RAM: 0	NVM: 88 RAM: file size in bytes
Counter	NVM: 92 + counter size in bytes RAM: 0	NVM: 88 RAM: 16
PCR	NVM: 176 RAM: 0	NVM: 140 RAM: 32
UserID	NVM: 100 + userID length in bytes RAM: 0	Not Applicable

8.2 Crypto Objects

Table 289. Crypto Object memory

Object Type	Object sub-type	NVM memory [bytes]	transient memory [bytes]
Digest	DIGEST_SHA	108	112
Digest	DIGEST_SHA224	108	112
Digest	DIGEST_SHA256	108	128
Digest	DIGEST_SHA384	108	208
Digest	DIGEST_SHA512	108	224
Cipher	DES_CBC_NOPAD	116	32
Cipher	DES_CBC_ISO9797_M1	116	16
Cipher	DES_CBC_ISO9797_M2	116	32
Cipher	DES_CBC_PKCS5	116	16
Cipher	DES_ECB_NOPAD	116	16
Cipher	DES_ECB_ISO9797_M1	116	16
Cipher	DES_ECB_ISO9797_M2	116	16
Cipher	DES_ECB_PKCS5	116	0
Cipher	AES_ECB_NOPAD	116	32
Cipher	AES_CBC_NOPAD	116	32
Cipher	AES_CBC_ISO9797_M1	116	48
Cipher	AES_CBC_ISO9797_M2	116	32

Table 289. Crypto Object memory *...continued*

Object Type	Object sub-type	NVM memory [bytes]	transient memory [bytes]
Cipher	AES_CBC_PKCS5	116	32
Cipher	AES_CTR	116	32
Signature	HMAC_SHA1	112	224
Signature	HMAC_SHA256	112	288
Signature	HMAC_SHA384	112	416
Signature	HMAC_SHA512	112	544
Signature	CMAC_128	116	32

•

9 Abbreviations

AES	Advanced Encryption Standard
API	Application Programming Interface
APDU	Application Protocol Data Unit
CLA	Class
DES	Data Encryption Standard
EC	Elliptic Curve
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie Hellman
ECKA	Elliptic Curve Key Agreement
FIPS	Federal Information Processing Standard
GCM	Gallois Counter Mode
GMAC	Gallois Counter Mode Message Authentication Code
HKDF	HMAC-based Key Derivation Function
I2C	Inter-Integrated Circuit
INS	Instruction
IoT	Internet of Things
KDF	Key Derivation Function
MAC	Message Authentication Code
PBKDF	Password Based Key Derivation Function
PCR	Platform Configuration Register
PRF	Pseudo Random Function
PSK	Pre Shared Key
Rev	Revision
RSA	Rivest Shamir Adleman
SCP	Secure Channel Protocol
TLS	Transport Layer Protocol
TLV	Tag Length Value

10 References

[1] **[ISO7816-4]** — ISO/IEC 7816-4:2013
Identification cards -- Integrated circuit cards -- Part 4: Organization, security and commands for interchange
https://www.iso.org/standard/54550.html

[2] **[SCP03]** — GlobalPlatform Card Technology
Secure Channel Protocol 03
Card Specification v 2.2 – Amendment D
Version 1.1.1
https://globalplatform.org/wp-content/uploads/2014/07/GPC_2.2_D_SCP03_v1.1.1.pdf

[3] **[IEEE-P1363]** — 1363-2000 - IEEE Standard Specification for Public-Key Cryptography
IEEE
29 Aug. 2000
https://ieeexplore.ieee.org/document/891000

[4] **[RFC3394]** — Advanced Encryption Standard (AES) Key Wrap Algorithm
Network Working Group
September 2002
https://tools.ietf.org/html/rfc3394

[5] **[Errata]** — SE050 Errata sheet
Rev. 1.0 — 14 October 2020
https://www.nxp.com/docs/en/errata/SE050_Erratasheet.pdf

11 Legal information

11.1 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.

11.2 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.

Limiting values — Stress above one or more limiting values (as defined in the Absolute Maximum Ratings System of IEC 60134) will cause permanent damage to the device. Limiting values are stress ratings only and (proper) operation of the device at these or any other conditions above those given in the Recommended operating conditions section (if present) or the Characteristics sections of this document is not warranted. Constant or repeated exposure to limiting values will permanently and irreversibly affect the quality and reliability of the device.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at http://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.

No offer to sell or license — Nothing in this document may be interpreted or construed as an offer to sell products that is open for acceptance or the grant, conveyance or implication of any license under any copyrights, patents or other industrial or intellectual property rights.

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.

Translations — A non-English (translated) version of a 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 or documented vulnerabilities. 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.

11.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are the property of their respective owners.

JCOP — is a trademark of NXP B.V.

NXP — wordmark and logo are trademarks of NXP B.V.

EdgeLock — is a trademark of NXP B.V.

Tables

Figures

Contents

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 24 March 2021
Document identifier: AN12413