AN13187

A5000 Authentication Application APDU Specification

Rev. 1.1 — 28 March 2022 | Application note

Document information

Information	Content
Keywords	A5000 Auth Application, Internet of Things, Secure authenticator
Abstract	This document provides the API description of the EdgeLock A5000 secure authenticator.

Revision history

Revision history

Rev	Date	Description
1.1	2022-03-28	Initial version
1.0	2021-10-29	Draft version

1 Introduction

1.1 Context

A5000 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 A5000 through an I²C interface (with the host controller being the I2C controller and the A5000 being the target).

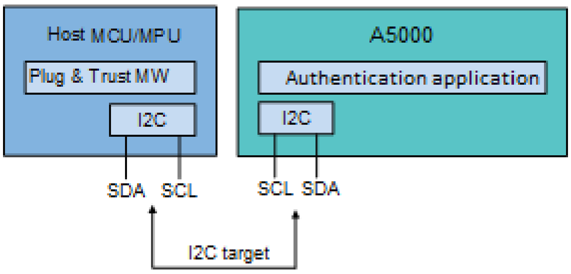

Figure 1. A5000 solution block diagram

The A5000 provides a wide range of (cryptographic) possibilities. Note that users need to be aware of the cryptographic principles when using the functionality of A5000 for the intended use cases.

Additional guidance is available in [UserGuidelines], each variant comes with a dedicated UGM.

2 A5000 architecture

2.1 Security Domain layout

NXP is in control of the Supplementary Security Domain (SSD) which holds the A5000 Authentication Application.

2.2 Application

The instance AID for A5000 Authentication Application - pre-provisioned by NXP - is A00000039654530000000103000000000.

The Application version is 7.2.0.

The APDU buffer size is 270 bytes.

Internally, the A5000 Authentication Application is using a command and response buffer of 1024 bytes. Any command that does not specify specific limitations on input and output is restricted by this buffer size of 1024 bytes.

3 A5000 Authentication Application functionality overview

This section provides an overview of the functionalities of the A5000 Authentication Application.

3.1 Supported functionality

The A5000 Authentication Application supports:

- Generic module management
  - Lifecycle management
  - Session management
  - Timer functionality
  - Access control
  - Secure import/export of keys or files
- Application Secure Channel management
  - AESKey sessions
  - ECKey sessions
- Random number generation
- Key management (ECC, AES, 3DES, etc.): write, read, lock, delete
- Elliptic curve cryptographic operations
- AES modes: ECB, CBC, CTR, GCM, CCM
- Random Initialization Vector generation for AES mode CTR, GCM and CCM.
- 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
  - GMAC
- Key derivation functionality
  - HKDF
- Specific use case support
  - TLS PSK master secret calculation

3.2 A5000 Secure Objects

3.2.1 Classes

The A5000 has one base object type called *Secure Object*. A Secure Object can be derived to classes depicted in Figure 2:

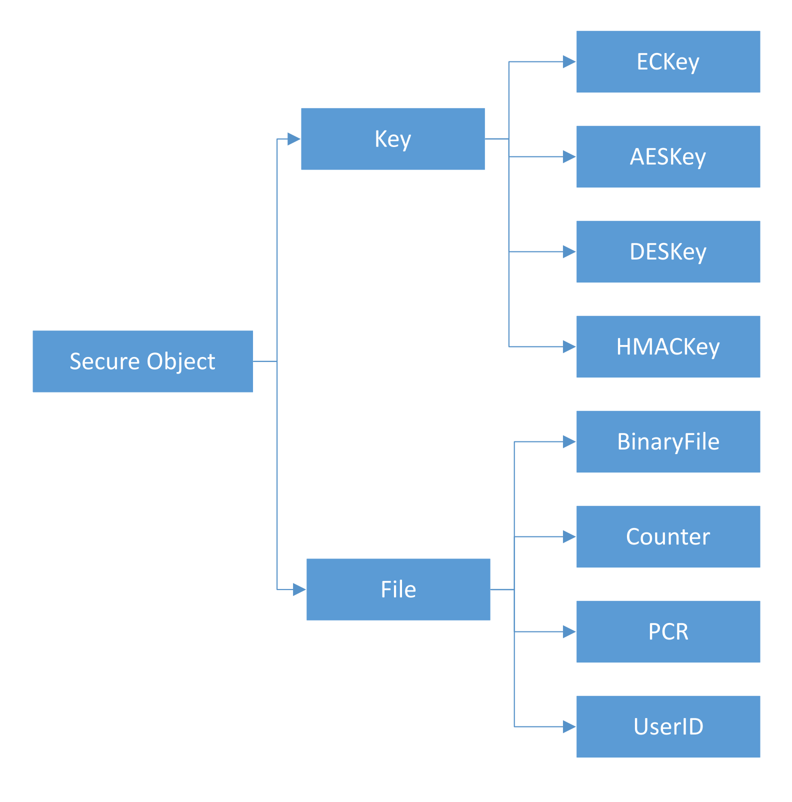

Figure 2. A5000 Secure Object structure

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 Section 4.3.15).

EC private keys are always stored in a ECPrivateKey object which size is exactly equal to EC curve bit size.

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.

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	Remarks
UNUSED	-			
NIST_P256	Y	32	65	

Table 1. Supported EC curves*...continued*

Name	Weiers trass	Private key byte length	Public key byte length	Remarks
NIST_P384	Y	48	97	

3.2.1.2 AESKey

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

3.2.1.3 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 A5000.

3.2.1.4 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.

3.2.1.5 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.

3.2.1.6 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.

3.2.1.7 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 Application (ClearOnDeselect); the initial value is restored once the Application 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.

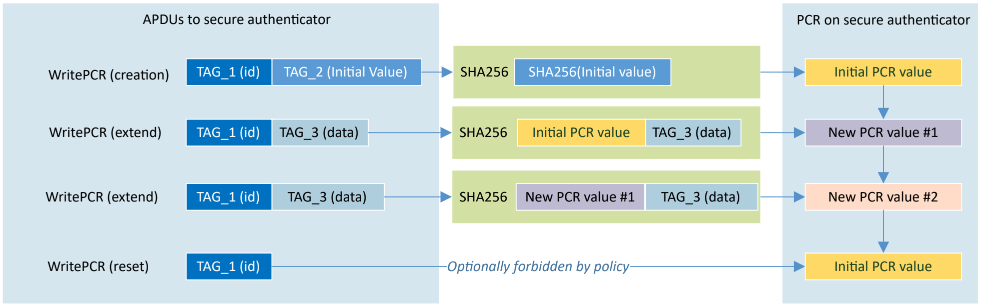

Figure 3. Example PCR sequence

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.

3.2.1.8 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 Application is deselected. Therefore, the objects survive a deselect, but the object contents will not survive. Keys will become invalid until they are set again, Files will have content being reset to all zero, except for PCRs, these will restore the initial value.

The Secure Object attributes are stored persistently, these remain unchanged after reselect.

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 accessed in sessions opened by this user. The same concept applies to the default session. Transient objects created within the default session can only be accessed in the context of the default session.

If a user tries to access a transient object created by another user, the Application rejects the command with SW ‘6985’.

Note that transient Secure Objects can be deleted by any user to which the policy POLICY_OBJ_ALLOW_DELETE is assigned, so deletion is not restricted to the session owner.

importExternalObject cannot be used for transient objects.

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. Authentication Objects are always persistent, transient Authentication Objects are not supported.

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 (described in UserID session).

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, AESKey (only 128 bit) or UserID.

For Authentication Objects of type ECKey, the public key component will be used on the secure authenticator (so either ECPublicKey or ECKeyPair objects) and only Weierstrass curves can be used for Authentication Objects, using others curves will lead to authentication failure.

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

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 (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 authenticator.

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 A5000. 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 for Authentication Objects and in Table 4 for non-Authentication Objects.

Table 3. Authentication Object attributes

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

Table 4. non-Authentication Objects

Attribute	Size (bytes)	Description
Object identifier	4	See Object identifiers
Object type	1	See Object type
Authentication indicator	1	See Authentication indicator
Minimum tag length for AEAD operations	2	See Minimum tag length for AEAD operations
Session Owner identifier	4	See Session Owner identifier
Minimum output length	2	See Minimum output length
Policy	Variable	See Policy
Origin	1	See Origin
Version	4	See Version

3.2.4.1 Object identifier

Each Secure Object is addressed using a 4-byte unique identifier. A Secure Object identifier is in the range of [0x00000001-0xFFFFFFFF], range of [0x7FFF0000-0x7FFFFFFF] is reserved for NXP. The identifier 0x00000000 is invalid and shall not be used. An object identifier is assigned when a new object is created.

The Table 5 below lists all the object identifiers which are reserved for specific purposes, such as Application configuration and management.

Table 5. Reserved file identifiers

Identifier	Description
0x00000000	Invalid object identifier
0x7FFF0200	RESERVED_ID_TRANSPORT
0x7FFF0201	RESERVED_ID_ECKEY_SESSION
0x7FFF0202	RESERVED_ID_EXTERNAL_IMPORT
0x7FFF0204	RESERVED_ID_FEATURE
0x7FFF0205	RESERVED_ID_FACTORY_RESET
0x7FFF0206	RESERVED_ID_UNIQUE_ID
0x7FFF0207	RESERVED_ID_PLATFORM_SCP
0x7FFF0209	RFU
0x7FFF020A	RESERVED_ID_RESTRICT

3.2.4.1.1 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.

3.2.4.1.1.1 RESERVED_ID_TRANSPORT

An authentication object which allows the user to switch SetLockState of the Application. The LockState defines whether the Application is transport locked or not.

3.2.4.1.1.2 RESERVED_ID_ECKEY_SESSION

A device unique key pair which contains the A5000 Key Agreement key pair in ECKey session context. See ECKey session.

3.2.4.1.1.3 RESERVED_ID_EXTERNAL_IMPORT

A device unique key pair which contains A5000 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. See Secure Object external import.

3.2.4.1.1.4 RESERVED_ID_FEATURE

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

3.2.4.1.1.5 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".

3.2.4.1.1.6 RESERVED_ID_UNIQUE_ID

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

3.2.4.1.1.7 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 Application session SCP.

3.2.4.1.1.8 RESERVED_ID_RESTRICT

An authentication object which grants access to the DisableObjectCreation command.

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.

Note that for ECKey objects, the returned type will always contain the curve ID in the returned value (so SecureObjectType will be > 0x20).

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 Minimum tag length for AEAD operations

The minimum AEAD tag length is a 2-byte value that defines the minimum tag length that is to be used in AEAD (encrypt and decrypt) operations when executing one of the commands:

- AEADInit
- AEADOneShot

This only applies to non Authentication Objects of type SymmKey.

Valid minimum tag lengths must be at least 4 bytes and at most 16 bytes, other values will not be accepted.

3.2.4.6 Session Owner identifier

“Owner” of the secure object; i.e., the 4-byte identifier of the session authentication object when the object has been created. Transient Secure Objects are bound to the Session Owner. They can only be accessed and used when the Session Owner attribute matches the Secure Object used to authenticate the current session.

Persistent Secure Objects are not bound to the Session Owner, these are fully controlled by policy management.

3.2.4.7 Minimum output length

The minimum output length of an HMACKey object that is required when executing one of the commands:

- HKDFExtractAndExpand:
- HKDFExpandOnly
- TLSCalculatePreMasterSecret

The minimum output length will only be enforced when the output is stored into a target object, the minimum does not apply when output to host is requested.

The minimum length can be defined in the WriteSymmKey command when creating a new HMACKey.

If the requested length is smaller than the minimum output length, an error will be returned and no data are stored in the target object.

3.2.4.8 Maximum authentication attempts

Maximum number of authentication attempts.

This value can be set when creating a new Secure Object as specified inTable 2.

The default value is 0, which means unlimited.

When this attribute is set (to a non-zero value), the Authentication Object cannot be used for authenticating anymore once the maximum number of authentication attempts is reached.

3.2.4.9 Policy

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

3.2.4.10 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). See Table 29.

Only Secure Objects of type ECKey can have ORIGIN_INTERNAL.

For Secure Objects of type File, the value is always set to ORIGIN_EXTERNAL or ORIGIN_PROVISIONED.

3.2.4.11 Version

Attribute that holds the version of the Secure Object. Default = 0. See Section 3.10 for details about versioning of Secure Objects.

3.2.5 Secure Object size

The Secure Object size will be reported in bytes:

- For EC keys: the size of the curve is returned, see ECKey for the exact size per curve.
- For AES/DES/HMAC keys, the key size is returned.
- For binary files: the total file size is returned, even when just a part of the object is read.
- For userIDs: the userID can be of any supported length, but 0 will be returned in all cases.
- For counters: the counter length is returned.
- For PCR: the PCR length is returned.

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 6). Also the size or other characteristics (e.g. EC curve) of the Secure Object cannot be modified on an existing object.

For any Secure Object op type Key (ECKey, 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.

When Secure Objects are used as target object to store the output of a C-APDU directly, the target Secure Object byte size must match exactly same size as the expected output size, else the APDU will fail to execute. This is applicable for the APDUs:

- ECDHGenerateSharedSecret: the target Secure Object must equal the length of the shared secret.
- ECPointMultiply: the target Secure Object must equal the length of an uncompressed EC point.
- HKDFExtractAndExpand: the target Secure Object must equal the length of the requested output length (see also minimum output length).
- HKDFExpandOnly: the target Secure Object must equal the length of the requested output length (see also minimum output length).
- TLSCalculatePreMasterSecret: the target Secure Object must equal the length of the TLS pre master secret (see also minimum output length).

Table 6. Secure Object Attribute updatability

Attribute	Updatable after object creation
Object identifier	N
Object type	N
Authentication attribute	N
Authentication attempt counter/tag length	Y (only the authentication attempt counter can reset when succesfully authenticating an ECKey session)

Table 6. Secure Object Attribute updatability*...continued*

Attribute	Updatable after object creation
Session Owner identifier	N
Maximum authentication attempts/ minimum output length	N
Policy	N
Origin	Y (only applies to Secure Objects of types ECKey, AESKey, DESKey and HMACKey, Counter and BinaryFile, see Object attributes)

3.2.7 Secure Object import/export

Transient Secure Objects of type AESKey, DESKey, ECCKey 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.

Exported credentials are always device individually encrypted and MAC'ed, so the import needs to be done on the same device as the export was triggered.

An object may only be imported if the SecureObject ID and type are the same as the exported object. Therefore, it is not possible to import if the corresponding Secure Object in the Application has been deleted.

Notes:

- The exported Secure Object key value is not deleted automatically.

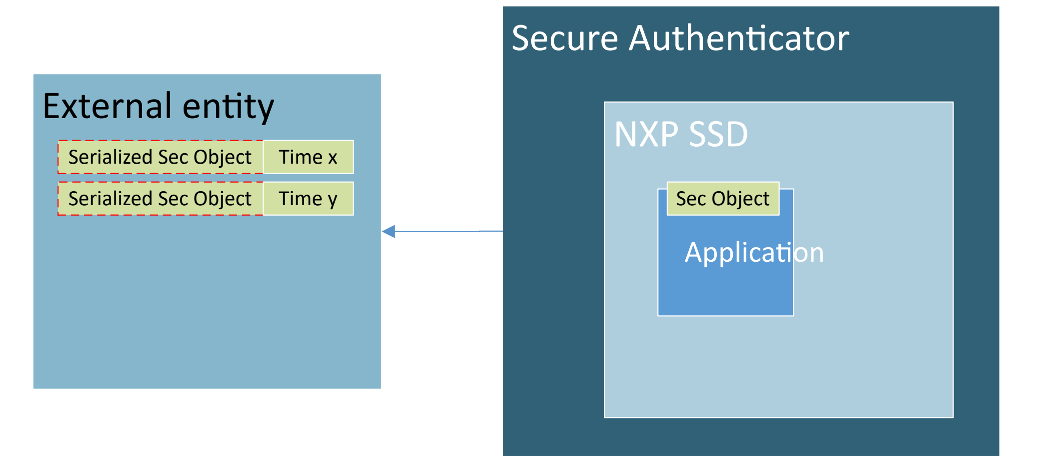

Figure 4. Secure Object import/export

3.2.8 Secure Object external import

Secure Objects can be imported into the A5000 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.

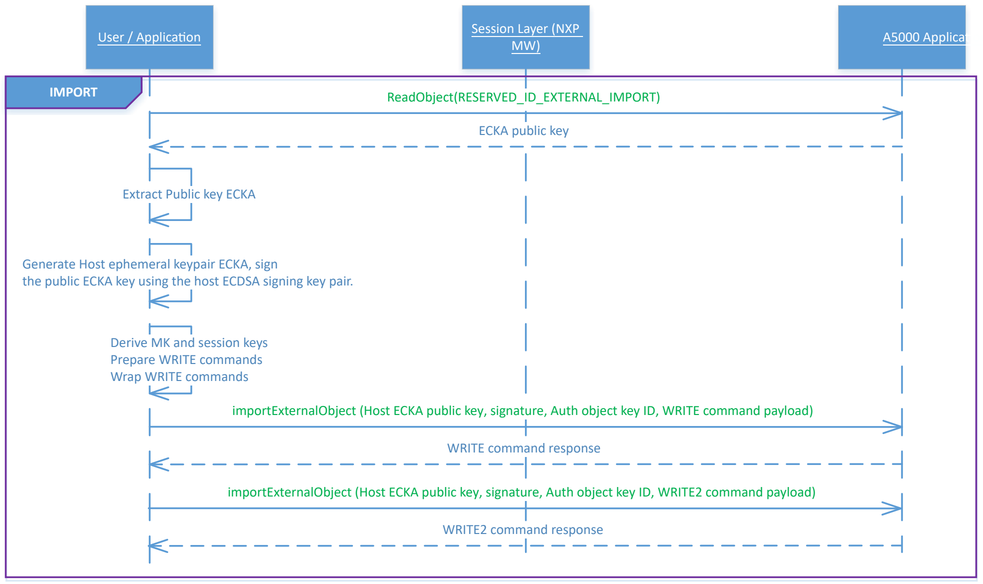
User / Application
Session Layer (NXP MW)
A5000 Applica
IMPORT
ReadObject(RESERVED_ID_EXTERNAL_IMPORT)
ECKA public key
Extract Public key ECKA
Generate Host ephemeral keypair ECKA, sign
the public ECKA key using the host ECDSA signing key pair.
Derive MK and session keys
Prepare WRITE commands
Wrap WRITE commands
importExternalObject (Host ECKA public key, signature, Auth object key ID, WRITE command payload)
WRITE command response
importExternalObject (Host ECKA public key, signature, Auth object key ID, WRITE2 command payload)
WRITE2 command response

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 an attested ReadObject command, passing identifier RESERVED_ID_EXTERNAL_IMPORT to get the public key from the device’s key pair. The attestation result needs to be checked for validity.
2. The user calls Section 4.6.2 with input:
   - the Application AID (e.g.A0000003965453000000010300000000)
   - the SCPparameters
     - 1-byte SCP identifier, must equal 0xAB
     - 2-byte SCP parameter, must equal 0x01 followed by 1-byte security level (which follows the GlobalPlatform security level definition, see: Table 8).
   - 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 Application 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 8.
- 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 concatenation of the MAC chaining value, the response APDU (which is empty) and the status word, using the RMAC session key,

Receipt = CMAC(RMAC session key, MCV | R-APDU | SW)

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

The ImportExternalObject commands can be replayed.

See ImportExternalObject for details.

3.3 Crypto Objects

3.3.1 Object types

A Crypto Object is an instance of a Cipher, Digest, Signature or AEAD 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 Using Crypto Objects

When a Crypto Object gets created, 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.

A Crypto Object can only be used (i.e. any crypto operation as well as managing the Crypto Object itself) by the creator of the Crypto Object. So a Crypto Object is bound to the Session Owner identifier attribute.

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.

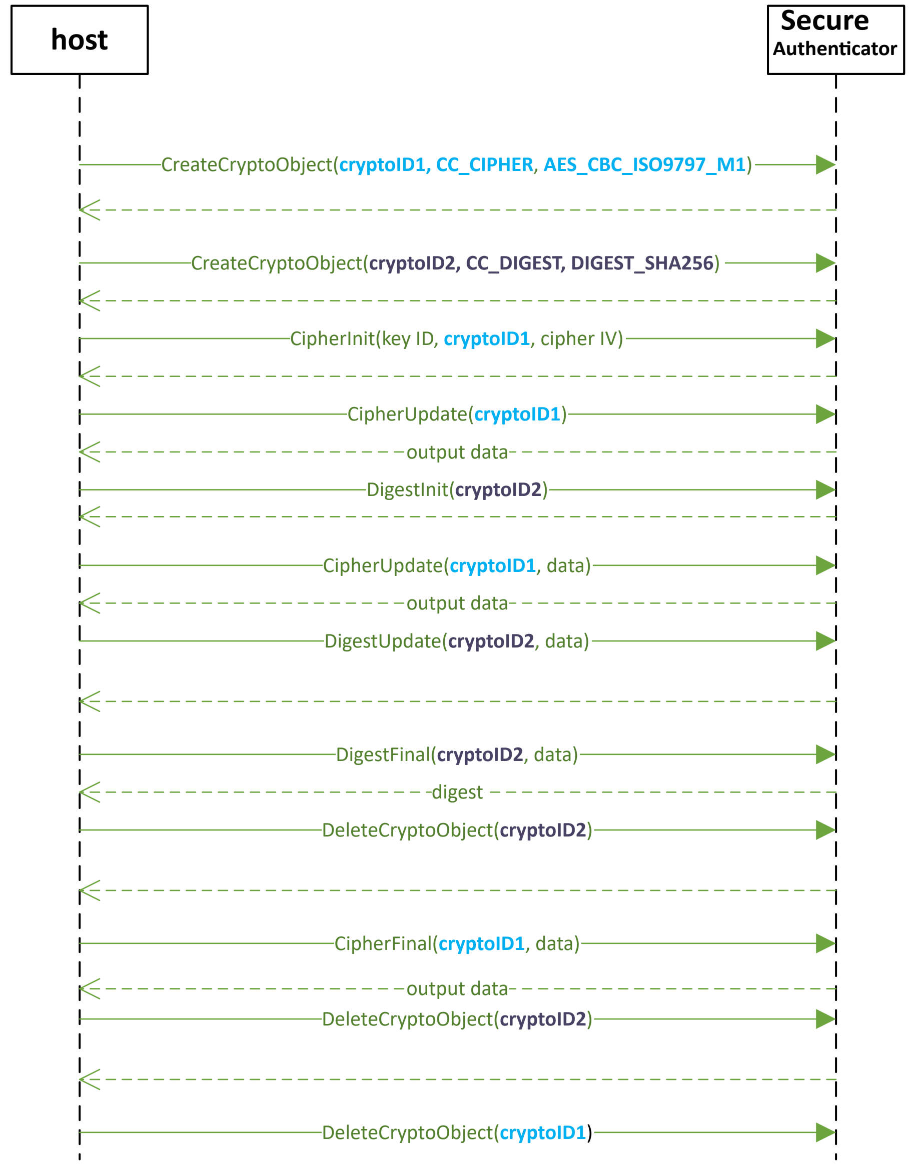
host
Secure Authenticator
CreateCryptoObject(cryptoID1, CC_CIPHER, AES_CBC_ISO9797_M1)
CreateCryptoObject(cryptoID2, CC_DIGEST, DIGEST_SHA256)
CipherInit(key ID, cryptoID1, cipher IV)
CipherUpdate(cryptoID1)
output data
DigestInit(cryptoID2)
CipherUpdate(cryptoID1, data)
output data
DigestUpdate(cryptoID2, data)
DigestFinal(cryptoID2, data)
digest
DeleteCryptoObject(cryptoID2)
CipherFinal(cryptoID1, data)
output data
DeleteCryptoObject(cryptoID2)
DeleteCryptoObject(cryptoID1)

Figure 6. Example Crypto Object usage

3.4 Supported Application features

An instance of the A5000 Authentication Application can be tuned to support specific functional blocks or features.

There are two bitmaps that defines Application features (ApplicationConfig) and feature bits (= extended feature bitmap), which can be set using SetApplicationFeatures. Note that these only reflect the Application functionality, depending on the operating system additional limitations might apply.

Note that users need to ensure both the features and extended features are filled properly.

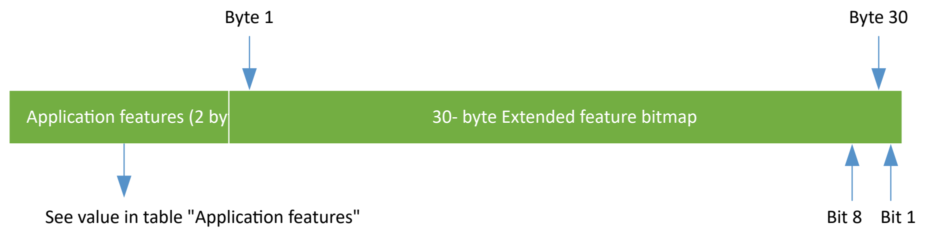
Byte 1
Byte 30
Application features (2 by
30- byte Extended feature bitmap
See value in table "Application features"
Bit 8
Bit 1

Figure 7. Setting Application features and extended features bitmap

Table 7. Application features

Feature	Value	Description
CONFIG_ECDSA_ECDH_ECDHE	0x0002	ECDSA and DH support (new key creation & key update)
CONFIG_RESERVED	0x0004	No functionality impact (reserved)
CONFIG_RESERVED	0x0008	No functionality impact (reserved)
CONFIG_HMAC	0x0010	Writing HMACKey objects (new key creation & key update)
CONFIG_RESERVED	0x0020	No functionality impact (reserved)
CONFIG_RESERVED	0x0040	No functionality impact (reserved)
CONFIG_AES	0x0080	Writing AESKey objects (new key creation & key update)
CONFIG_DES	0x0100	Writing DESKey objects (new key creation & key update)
CONFIG_TLS	0x0400	TLS Handshake support commands (crypto operations, see TLS handshake support)
CONFIG_RESERVED	0x0800	No functionality impact (reserved)
CONFIG_RESERVED	0x1000	No functionality impact (reserved)
CONFIG_RESERVED	0x2000	No functionality impact (reserved)
CONFIG_RESERVED	0x4000	No functionality impact (reserved)

3.5 Secure Channel Protocols

3.5.1 Multi-level SCP

The A5000 Authentication Application 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, see [2].

- **Application level SCP**: for end-to-end secure channel protection. The commands to set up a secure channel on Application 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 A5000 Authentication Application uses the Security Level definitions as defined in GlobalPlatform, (see Table 10-1 in [SCP03]) and as depicted in Table 8.

Table 8. Security Level

B8	B7	B6	B5	B4	B3	B2	B1	Meaning
-	-	-	-	-	-	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 A5000 Authentication Application allows to set up **Application sessions**. An Application session is an authenticated communication channel between the owner of an Authentication Object and the A5000 Authentication Application.

Commands can be sent to the A5000 Authentication Application either:

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

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

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

3.6.1 Session-less access

By default, the Application 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 A5000 use against multi-threaded behavior (as any user or thread can interfere at any moment).

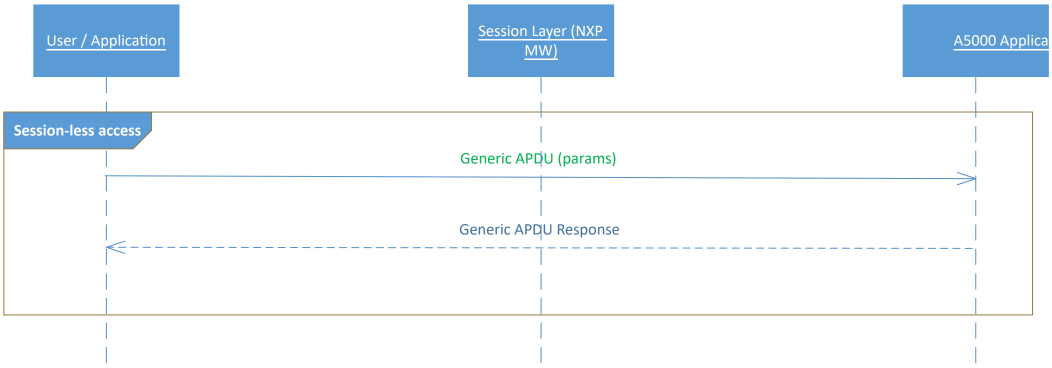

Figure 8. Session-less access

Note:

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

3.6.2 Application sessions

The following Application 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) Application 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.

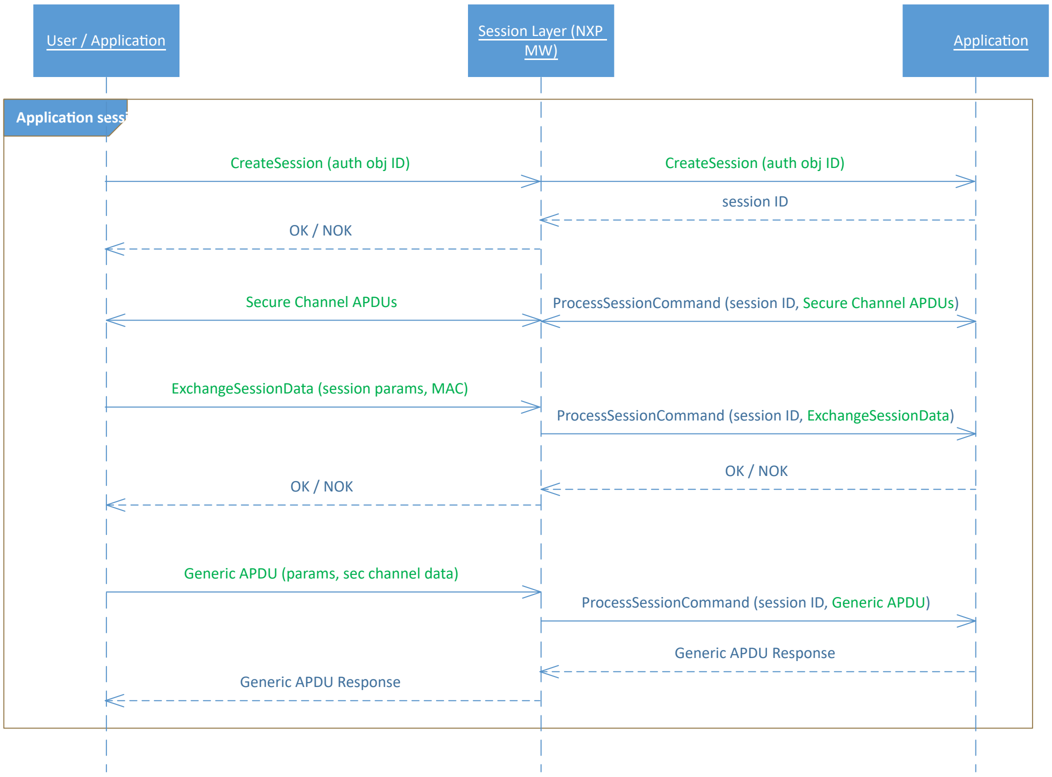
User / Application
Session Layer (NXP MW)
Application
Application sessi
CreateSession (auth obj ID)
CreateSession (auth obj ID)
session ID
OK / NOK
Secure Channel APDUs
ProcessSessionCommand (session ID, Secure Channel APDUs)
ExchangeSessionData (session params, MAC)
ProcessSessionCommand (session ID, ExchangeSessionData)
OK / NOK
OK / NOK
Generic APDU (params, sec channel data)
ProcessSessionCommand (session ID, Generic APDU)
Generic APDU Response
Generic APDU Response

Figure 9. Application 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 Application 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.

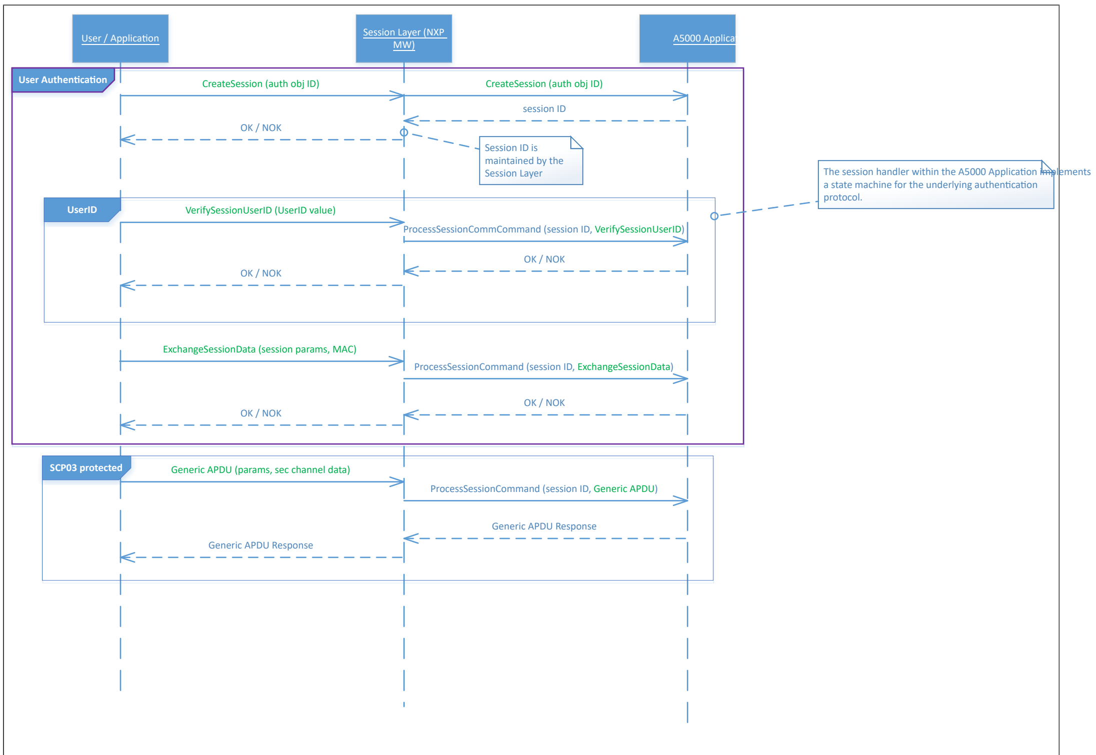
User / Application
Session Layer (NXP MW)
A5000 Applica
User Authentication
CreateSession (auth obj ID)
CreateSession (auth obj ID)
session ID
OK / NOK
Session ID is maintained by the Session Layer
The session handler within the A5000 Application implements a state machine for the underlying authentication protocol.
UserID
VerifySessionUserID (UserID value)
ProcessSessionCommCommand (session ID, VerifySessionUserID)
OK / NOK
OK / NOK
ExchangeSessionData (session params, MAC)
ProcessSessionCommand (session ID, ExchangeSessionData)
OK / NOK
OK / NOK
SCP03 protected
Generic APDU (params, sec channel data)
ProcessSessionCommand (session ID, Generic APDU)
Generic APDU Response
Generic APDU Response

Figure 10. 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.

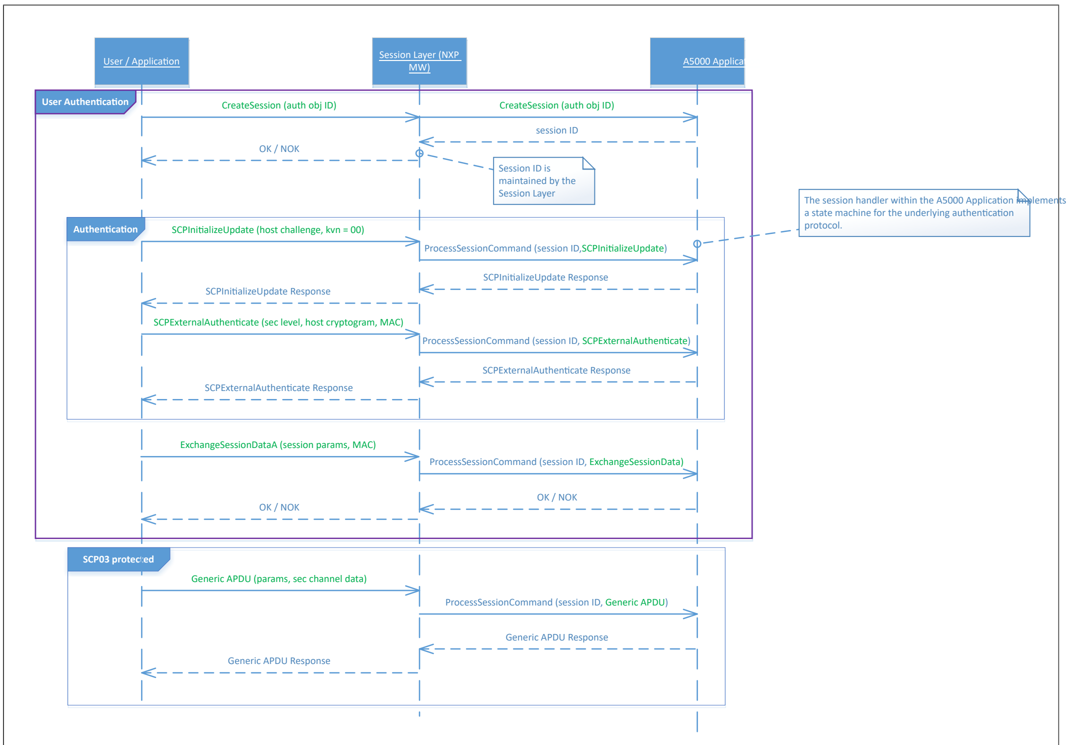
User / Application
Session Layer (NXP MW)
A5000 Applica
User Authentication
CreateSession (auth obj ID)
CreateSession (auth obj ID)
session ID
OK / NOK
Session ID is maintained by the Session Layer
The session handler within the A5000 Application implements a state machine for the underlying authentication protocol.
Authentication
SCPInitializeUpdate (host challenge, kvn = 00)
ProcessSessionCommand (session ID,SCPInitializeUpdate)
SCPInitializeUpdate Response
SCPInitializeUpdate Response
SCPExternalAuthenticate (sec level, host cryptogram, MAC)
ProcessSessionCommand (session ID, SCPExternalAuthenticate)
SCPExternalAuthenticate Response
SCPExternalAuthenticate Response
ExchangeSessionDataA (session params, MAC)
ProcessSessionCommand (session ID, ExchangeSessionData)
OK / NOK
OK / NOK
SCP03 protected
Generic APDU (params, sec channel data)
ProcessSessionCommand (session ID, Generic APDU)
Generic APDU Response
Generic APDU Response

Figure 11. Session creation using an AES key as authentication object

3.6.3.3 ECKey session

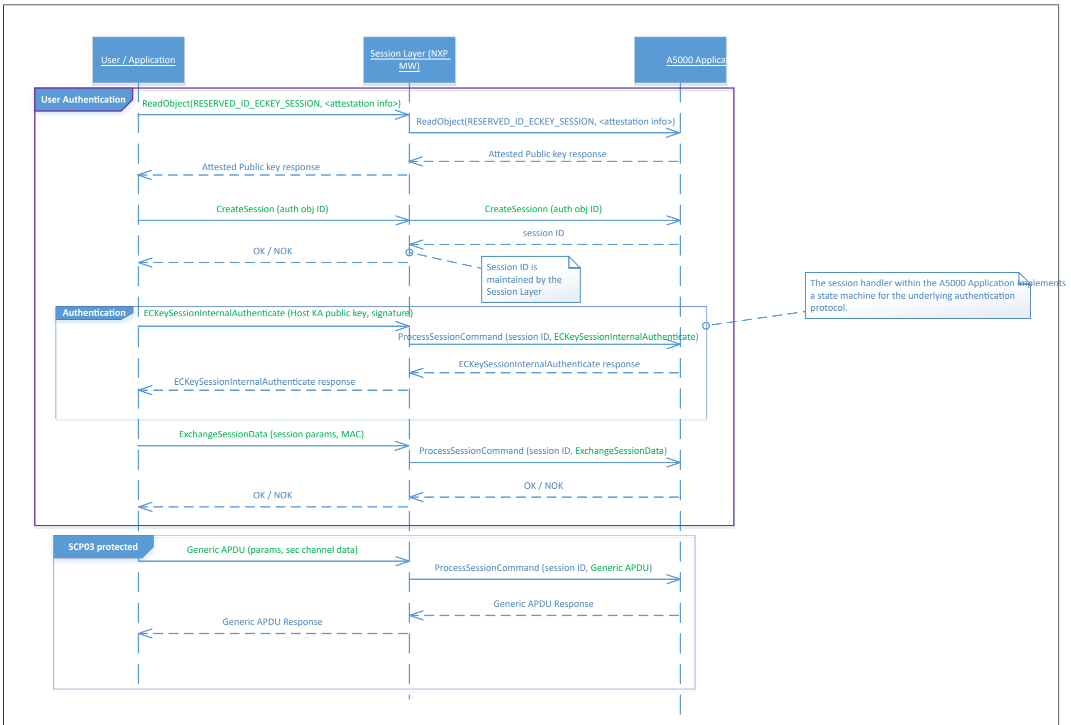
User / Application
Session Layer (NXP MW)
A5000 Applica
User Authentication
ReadObject(RESERVED_ID_ECKEY_SESSION, <attestation info>)
ReadObject(RESERVED_ID_ECKEY_SESSION, <attestation info>)
Attested Public key response
Attested Public key response
CreateSession (auth obj ID)
CreateSessionn (auth obj ID)
session ID
OK / NOK
Session ID is maintained by the Session Layer
The session handler within the A5000 Application implements a state machine for the underlying authentication protocol.
Authentication
ECKeySessionInternalAuthenticate (Host KA public key, signature)
ProcessSessionCommand (session ID, ECKeySessionInternalAuthenticate)
ECKeySessionInternalAuthenticate response
ECKeySessionInternalAuthenticate response
ExchangeSessionData (session params, MAC)
ProcessSessionCommand (session ID, ExchangeSessionData)
OK / NOK
OK / NOK
SCP03 protected
Generic APDU (params, sec channel data)
ProcessSessionCommand (session ID, Generic APDU)
Generic APDU Response
Generic APDU Response

Figure 12. Session creation using ECKey session as authentication mechanism.

An ECKey Session is established as follows:

1.

The user obtains the SE public key for import via an attested ReadObject command, passing identifier RESERVED_ID_ECKEY_SESSION to get the public key from the device's key pair. The attestation result needs to be checked for validity. This step only needs to be done for the first ECKey session setup. Any successive ECKey session setup can reuse the key requested initially.

2.

The user calls CreateSession with the desired authentication object ID (EC public key or EC Keypair) and receives a session ID.

3.

To prof the knowledge of the authentication object secret, the user calls ECKeySessionInternalAuthenticate with input:

- the Application AID (e.g. A00000039654530000000103000000000)
- 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 8). 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); for each ECKey session setup, this key must be a unique key, so this should be the public key of an ephemeral key pair.
- host public key curve identifier (must be 0x03 (= NIST_P256))
- ASN.1 signature over the TLV with tags 0xA6 and 0x7F49 (using the Host Private key).

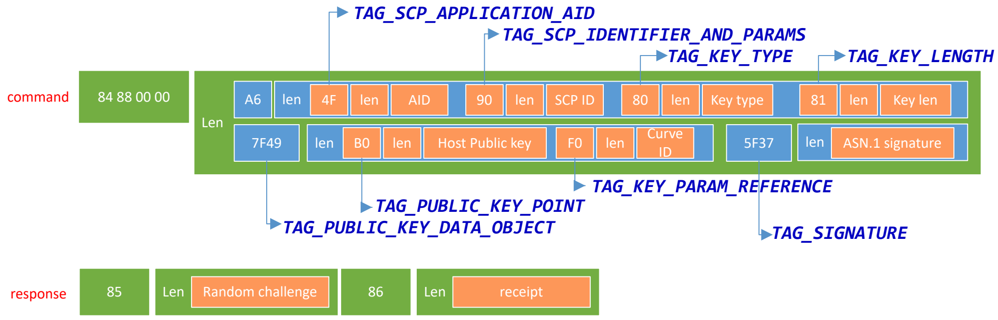

Figure 13. ECKeySessionInternalAuthenticate

The Application 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 (see Supported EC Curves).
- 16-byte random generated by the A5000 as returned in the response of the ECKeySessionInternalAuthenticate command.

- 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.

4.

Commands can be exchanged in the protected session.

When secure objects have a policy specified for the authentication object ID (as passed via CreateSession) the respective Access Rules are active within this session.

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 9

Table 9. 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 10.

Table 10. Policy set

Field	Length	Description
Policy	9-53	First policy
Policy	9-53	Second policy
…	…	…

3.7.1.2 Policy

Each policy is structured according to Table 11 and Table 12.

Table 11. 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.	M
Access rules (AR)	4, 8, 12, 40, 44 or 48 bytes	See Section 3.7.1.3	M

Table 12. Access Rule structure

	Field	Length	Description	M / O/ C
	AR Header	4	Access rules header of fixed size	M
	AR Extension	0, 4, 8, 36, 40 or 44	Optional access rules extension	C

Notes:

- The Authentication Object ID defines the Authentication Object to which the access rules apply. When the value is 0x00000000, the access rules apply to the default session, which can be opened by anybody.
  E.g.: 08**00000000**0014000008**11111111**18000000 assigns the Access Rule 00140000 to the default session. However, these access rules are not inherited by sessions with the Authentication Object with identifier 0x11111111.
- Transient objects are bound to the creating session, to prevent interference by other sessions. Access rules set for transient objects which are not related to the current session do not have an effect, as the transient object will not be accessible from those other sessions.
- For a single policy set, the policies need to contain unique Authentication Object IDs: a certain Authentication Object ID cannot be present multiple times in the same policy set.
  E.g. 08**00000000**0014000008**00000000**18000000 will not work and should be constructed as 08**00000000**18140000.
- If users do not set a specific Policy Set, the default policies apply to the object. The Default Policy applies to any session.

3.7.1.3 Access Rule

An access rule defines which operations are allowed on an object. As defined in Table 11 and Table 12 , an access rule contains a mandatory 4-byte header and an optional extension of up to 44 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 ‘4FFFF000’ shall have value ‘00112233445566778899AABBCCDDEEFF00112233445566778899AABBCCDDEEFF’ (POLICY_OBJ_REQUIRE_PCR_VALUE)

The above example policy would be coded as follows:

- Policy length: ‘2C’ (44 bytes total)
- Access rule header: ‘00210000’ (POLICY_OBJ_ALLOW_READ | POLICY_OBJ_REQUIRE_PCR_VALUE)
- Access rule extension: ‘4FFFF00000112233445566778899AABBCCDDEEFF00112233445566778899AABBCCDDEEFF’

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.

Table 13 defines which access rules are allowed for each object class, as defined in Classes.

Table 13. Policy validation per object type

Object class	Applicable access rules
Policies applicable to all Secure Objects: • Symmetric Key Objects • Asymmetric Key Objects • BinaryFile, Counter, PCR or UserID Secure Objects	POLICY_OBJ_FORBID_ALL POLICY_OBJ_ALLOW_READ POLICY_OBJ_ALLOW_WRITE POLICY_OBJ_ALLOW_DELETE POLICY_OBJ_REQUIRE_SM POLICY_OBJ_REQUIRE_PCR_VALUE
Additional policies applicable to Symmetric Key Objects (policies specific to Symmetric Key Objects are put in *italic*)	POLICY_OBJ_ALLOW_SIGN POLICY_OBJ_ALLOW_VERIFY POLICY_OBJ_ALLOW_ENC POLICY_OBJ_ALLOW_DEC POLICY_OBJ_ALLOW_IMPORT_EXPORT POLICY_OBJ_ALLOW_DERIVED_INPUT POLICY_OBJ_FORBID_DERIVED_OUTPUT *POLICY_OBJ_ALLOW_TLS_KDF* *POLICY_OBJ_ALLOW_TLS_KDF_EXT_RANDOM* *POLICY_OBJ_ALLOW_TLS_PMS* *POLICY_OBJ_ALLOW_HKDF* *POLICY_OBJ_ALLOW_RFC3394_UNWRAP* *POLICY_OBJ_FORBID_EXTERNAL_IV* *POLICY_OBJ_ALLOW_USAGE_AS_HMAC_PEPPER*
Additional policies applicable to Asymmetric Key Objects (policies specific to Asymmetric Key Objects are put in *italic*)	POLICY_OBJ_ALLOW_SIGN POLICY_OBJ_ALLOW_VERIFY POLICY_OBJ_ALLOW_ENC POLICY_OBJ_ALLOW_DEC POLICY_OBJ_ALLOW_IMPORT_EXPORT POLICY_OBJ_ALLOW_DERIVED_INPUT POLICY_OBJ_FORBID_DERIVED_OUTPUT *POLICY_OBJ_ALLOW_GEN* *POLICY_OBJ_ALLOW_KA* *POLICY_OBJ_ALLOW_ATTESTATION* *POLICY_OBJ_INTERNAL_SIGN*

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 14.

Table 14. 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
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. Default policies are enabled only for ease of use; users must define a (non-default) policy for each Secure Object based on the security requirements for the product.

Table 15. Default object policies

Object type	Default policy
Authentication Object	Maximum attempts: unlimited. Applied policies: POLICY_OBJ_ALLOW_READ
Non-Authentication Object all classes (policies applicable to all classes defined below, regardless of their type) [any Secure Object type]	POLICY_OBJ_ALLOW_READ POLICY_OBJ_ALLOW_WRITE POLICY_OBJ_ALLOW_DELETE
Non-Authentication Object Symmetric key [AES, DES, HMAC]	POLICY_OBJ_ALLOW_SIGN POLICY_OBJ_ALLOW_VERIFY POLICY_OBJ_ALLOW_ENC POLICY_OBJ_ALLOW_DEC POLICY_OBJ_ALLOW_IMPORT_EXPORT *POLICY_OBJ_ALLOW_HKDF*
Non-Authentication Object Asymmetric key	POLICY_OBJ_ALLOW_SIGN POLICY_OBJ_ALLOW_VERIFY POLICY_OBJ_ALLOW_ENC POLICY_OBJ_ALLOW_DEC POLICY_OBJ_ALLOW_IMPORT_EXPORT *POLICY_OBJ_ALLOW_GEN* *POLICY_OBJ_ALLOW_KA*

Table 16. Default session policies

Object type	Default policy
Session	No maximum number of APDU or command limitations Session refresh is not allowed

3.7.4 Authentication Object policies

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

- POLICY_OBJ_ALLOW_READ
- POLICY_OBJ_ALLOW_WRITE
- POLICY_OBJ_ALLOW_DELETE

Some policies can be set, but do not have effect on Authentication Objects, e.g.:

- POLICY_OBJ_REQUIRE_SM
- POLICY_OBJ_REQUIRE_PCR_VALUE
- POLICY_OBJ_FORBID_ALL

3.7.5 Policy check

When a Secure Object exists and a new Write command is sent to update the value of the object, the user can insert the policy of the object into the C-APDU to ensure that the object is only written when the given policy equals the existing policy. The policy given in the TLV[TAG_POLICY_CHECK] must match exactly with the policy that is present on that Secure Object, else the C-APDU will be rejected.

This only applies when the value of the object is given as input for the C-APDU, it is ignored when there is no input argument (e.g. when using a Write command to generate a key inside the A5000).

3.7.6 Policy usage

This chapter will give more detailed information on certain policies.

3.7.6.1 POLICY_OBJ_FORBID_ALL

POLICY_OBJ_FORBID_ALL can be applied to prevent a particular session owner to have access to a Secure Object.

3.7.6.2 POLICY_OBJ_FORBID_DERIVED_OUTPUT

When a function allows optionally output to be stored into a target object instead of returning via the R-APDU, the POLICY_OBJ_FORBID_DERIVED_OUTPUT can be applied on the source object to prevent output being returned to the host and as such mandate the use of a target object. Functions that do not store output in a target object would also block output to host in case this policy is set.

Applicable functions:

- ECDSASign
- ECDHGenerateSharedSecret
- ECPointMultiply
- CipherInit
- CipherOneShot
- AEADInit
- AEADOneShot
- MACInit (only when generating a MAC value)
- MACOneShot (only when generating a MAC value)
- HKDFExtractThenExpand

- HKDFExpandOnly
- TLSPerformPRF

Note that target objects are not implicitely created, so these must be created upfront by calling WriteSecureObject.

3.7.6.3 POLICY_OBJ_ALLOW_DERIVED_INPUT

When a target object is derived from a source object, the POLICY_OBJ_ALLOW_DERIVED_INPUT restricts the target object to be derived from a single source object.

The policy takes a 4-byte extension pointing to the source object that needs to be used in the key derivation. For any other source object, an error would be returned.

Applicable functions:

- ECDHGenerateSharedSecret
- ECPointMultiply
- HKDFExtractThenExpand
- HKDFExpandOnly
- TLSCalculatePreMasterSecret

This policy is overruled by the POLICY_OBJ_ALLOW_WRITE: When POLICY_OBJ_ALLOW_WRITE is applied to the target object, any source object would be allowed to derive into that target object (in case the user is allowed by that policy), even if POLICY_OBJ_ALLOW_DERIVED_INPUT is applied on the object.

3.7.6.4 POLICY_OBJ_FORBID_EXTERNAL_IV

POLICY_OBJ_FORBID_EXTERNAL_IV can be applied to enforce internal IV generation for specific commands. This policy, together with POLICY_OBJ_ALLOW_ENC, denies input of an external IV and limits the use of the Secure Object to the following algorithms:

- AES CTR
- AES CCM
- AES GCM/GMAC

Applicable functions:

- CipherInit
- CipherOneShot
- AEADInit
- AEADOneShot

If the policy is applied to a Secure Object, it will only allow encryption, decryption is blocked -regardless of whether POLICY_OBJ_ALLOW_DEC is applied as well or not-.

3.8 Lifecycle management

The Application 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 17.

Table 17. Commands allowed in Inactive state

Command	Remark
GetVersion	
ReadState	
ReadObject	Only object with identifier RESERVED_ID_UNIQUE_ID can be read.
GetRandom	
CreateSession	

The Application can be set to Inactive state calling SetLockState.

Unlocking the Application 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 strongly monotonic, i.e. any read will return an increased value (excl. wrap around).

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

3.10 Secure Object versioning

For the following Secure Objects, a *version* can be passed as input parameter:

- ECKey objects
- SymmKey objects
- BinaryFile objects

By default, the version of an object is 0.

If versioning is required, the user can pass a non-zero value as version of the object.

In that case, any further write attempt to the object must include a version number that is equal to or higher than the stored version, else the command will be rejected. Note that a write attempt means either key generation (on chip), key insertion or importing an external object.

For Secure Objects written in multiple APDUs (e.g. Binary Files), each APDU must contain the same version -in case a version is present-.

If the version is 0, no additional NVM writes are done besides the Secure Object value update itself.

The maximum version is 0x7FFFFFFF. When the maximum is reached, no further write attempt is possible.

3.11 Disable Secure Object creation

It is possible to prevent creation or creation + update of Secure Objects on A5000, either persistent or transient, by calling DisableSecureObjectCreation.

The user can choose to permanently disable the creation which is irreversible, or transiently disable creation which remains valid until next deselect.

The user can choose to prevent creation of additional objects (= least restrictive) or to prevent creation of additional objects and update of existing objects (= most restrictive).

The feature is protected by the RESERVED_ID_RESTRICT.

3.12 Mandate of platform SCP channel

The A5000 allows to mandate the use of platform SCP by calling SetPlatformSCPRequest. When enabled, users must be authenticated to the platform with highest security level (i.e. C_DECRYPTION/C_MAC and R_ENCRYPTION/R_MAC), else the command will be rejected by the Authentication Application.

Exceptions that will not be rejected are:

- Select
- GetVersion
- ReadState

3.13 Garbage collection

The A5000 supports garbage collection to clean up memory.

Garbage collection is triggered when either of these items is deleted:

- Secure Object -only the Secure OBject that is deleted is cleaned up, no linked object etc.-
- Crypto Object
- EC curve

When garbage collection is triggered, the garbage collection will run when the next incoming APDU is handled. This might have a negative impact on the performance of that C-APDU handling.

4 A5000 APDU interface

4.1 APDU Format

A5000 Authentication Application 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.

When the response would contain more than 256 bytes, the C-APDU must be an extended length APDU, i.e. the Le field must be 3 bytes long, else the Application would respond SW_CONDITIONS_NOT_SATISFIED.

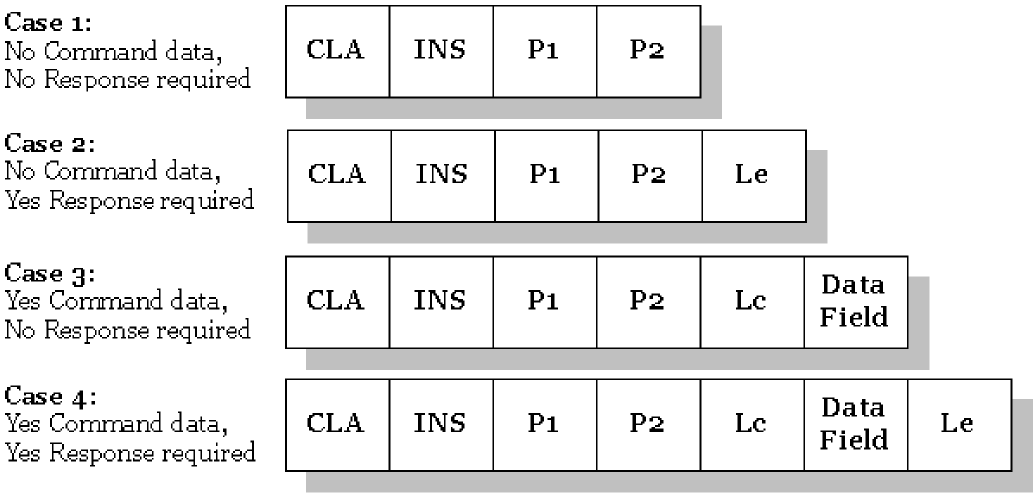

Figure 14. APDU format

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 Application. 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 E.2: 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 E.3: 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 18. Error codes

Name	Value	Description
SW_NO_ERROR	0x9000	No Error
SW_WRONG_LENGTH	0x6700	Wrong length (e.g. C-APDU does not fit into APDU buffer)
SW_CONDITIONS_NOT_SATISFIED	0x6985	Conditions not satisfied
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
SW_FILE_FULL	0x6A84	Not enough memory space available (either transient or persistent memory).

4.3.2 General

Table 19. General constants

Name	Value	Description
MAX_NUMBER_OF_SESSIONS	2	Maximum number of simultaneous Applica tion sessions (excl. session-less access)

4.3.3 Instruction

Table 20. Instruction mask constants

Name	Value	Description
INS_MASK_INS_CHAR	0xF0	4 MSBit for instruction characteristics.
INS_MASK_INSTRUCTION	0x0F	4 LSBit for instruction

Table 21. 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 22. 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 23. 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 24. 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 25. P1Cred constants

Name	Value
P1_DEFAULT	0x00
P1_EC	0x01
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
P1_CIPHER	0x0E

Table 25. P1Cred constants*...continued*

Name	Value
P1_TLS	0x0F
P1_CRYPTO_OBJ	0x10
P1_AEAD	0x11

4.3.5 P2 parameter

Table 26. 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_VERSION_EXT	0x21
P2_MEMORY	0x22
P2_LIST	0x25
P2_TYPE	0x26

Table 26. P2 constants*...continued*

Name	Value
P2_EXIST	0x27
P2_DELETE_OBJECT	0x28
P2_DELETE_ALL	0x2A
P2_SESSION_USERID	0x2C
P2_HKDF	0x2D
P2_HKDF_EXPAND_ONLY	0x2F
P2_MAC	0x32
P2_UNLOCK_CHALLENGE	0x33
P2_CURVE_LIST	0x34
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
P2_SCP	0x52
P2_AUTH_FIRST_PART1	0x53

Table 26. P2 constants*...continued*

Name	Value
P2_AUTH_NONFIRST_PART1	0x54
P2_CM_COMMAND	0x55
P2_RESTRICT	0x57
P2_SANITY	0x58
P2_DH_REVERSE	0x59
P2_PRF_BOTH	0x5A
P2_STATE	0x5B

4.3.6 SecureObject type

Note: *TYPE_EC_KEY_PAIR, TYPE_EC_PRIV_KEY and TYPE_EC_PUB_KEY are not returned, the curve will always be included for respectively EC key pairs, EC private keys or EC public keys.*

Table 27. SecureObjectType constants

Name	Value
TYPE_EC_KEY_PAIR	0x01
TYPE_EC_PRIV_KEY	0x02
TYPE_EC_PUB_KEY	0x03
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 28. 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 29. Origin constants

Name	Value	Description
ORIGIN_EXTERNAL	0x01	Generated outside the module.

Table 29. Origin constants*...continued*

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

4.3.9 TLV tags

Table 30. 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_POLICY_CHECK	0x15
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
TAG_11	0x4B
TAG_TS	0x4F
TAG_ATT_SIG	0x52

4.3.10 ECSignatureAlgo

Table 31. ECSignatureAlgo

Name	Value	Description
SIG_ECDSA_PLAIN	0x09	NOT SUPPORTED
SIG_ECDSA_SHA_256	0x21	ECDSA with a SHA256 digest as input.
SIG_ECDSA_SHA_384	0x22	ECDSA with a SHA384 digest as input.

4.3.11 ECDHAlgo

Table 32. ECDHAlgo

Name	Value	Description
EC_SVDP_DH	0x01	Generates the SHA1 of the X coordinate.
EC_SVDP_DH_PLAIN	0x03	Generates the X coordinate

4.3.12 ECPMAlgo

Table 33. ECPMAlgo

Name	Value	Description
EC_SVDP_DH_PLAIN_XY	0x06	Generates the uncompressed EC point XY.

4.3.13 DigestMode

Table 34. DigestMode constants

Name	Value
DIGEST_NO_HASH	0x00
DIGEST_SHA256	0x04
DIGEST_SHA384	0x05

4.3.14 MACAlgo

Table 35. MACAlgo constants

Name	Value	Description
HMAC_SHA256	0x19	
HMAC_SHA384	0x1A	
CMAC128	0x31	
DES_CMAC8	0x7A	Only available in DigestOneShot.

4.3.15 ECCurve

Table 36. ECCurve constants

Name	Curve ID	Weierstrass
UNUSED	0x00	-
NIST_P256	0x03	Y
NIST_P384	0x04	Y

4.3.16 ECCurveParam

Table 37. ECCurveParam constants

Name	Value
CURVE_PARAM_A	0x01
CURVE_PARAM_B	0x02

Table 37. ECCurveParam constants*...continued*

Name	Value
CURVE_PARAM_G	0x04
CURVE_PARAM_N	0x08
CURVE_PARAM_PRIME	0x10

4.3.17 CipherMode

Table 38. CipherMode constants

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

4.3.18 AEADMode

Table 39. AEADMode

Name	Value	Description
AES_GCM	0xB0	AES GCM/GMAC operations
AES_CCM	0xF4	AES CCM operations

4.3.19 AttestationAlgo

AttestationAlgo is ECSignatureAlgo .

4.3.20 LockIndicator

Table 40. LockIndicator constants

Name	Value
TRANSIENT_LOCK	0x01
PERSISTENT_LOCK	0x02

4.3.21 LockState

Table 41. LockState constants

Name	Value
LOCKED	0x01
UNLOCKED	Any except 0x01

4.3.22 RestrictMode

Table 42. RestrictMode constants

Name	Value
RESTRICT_NEW	0x01
RESTRICT_ALL	0x02

4.3.23 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
CC_AEAD	0x04	For AEADInit/AEADUpdate/AEADFinal

4.3.24 Result

Table 44. Result constants

Name	Value
RESULT_SUCCESS	0x01
RESULT_FAILURE	0x02

4.3.25 TransientIndicator

Table 45. TransientIndicator constants

Name	Value
PERSISTENT	0x01
TRANSIENT	0x02

4.3.26 SetIndicator

Table 46. SetIndicator constants

Name	Value
NOT_SET	0x01
SET	0x02

4.3.27 MoreIndicator

Table 47. MoreIndicator constants

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

4.3.28 HealthCheckMode

Table 48. HealthCheckMode constants

Name	Value	Description
HCM_ CODE_SIGNATURE	0xFE01	Performs ROM integrity checks. When the test fails, the chip triggers the attack counter and the chip will reset.
HCM_DYNAMIC_FLASH_ INTEGRITY	0xFD02	Performs flash integrity tests. When the test fails, the chip triggers the attack counter and the chip will reset.
HCM_SHIELDING	0xFC03	Performs tests on the active shield protection of the hardware. When the test fails, the chip triggers the attack counter and the chip will reset.
HCM_SENSOR	0xFB04	Performs self-tests on hardware sensors and reports the status.
HCM_SFR_CHECK	0xFA05	Performs self-tests on the hardware registers. When the test fails, the chip triggers the attack counter and the chip will reset.

4.3.29 PlatformSCPRequest

Table 49. PlatformSCPRequest constants

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

4.3.30 CryptoObject

A CryptoObject is a 2-byte value consisting of a CryptoContext in MSB and one of the following in LSB:

4.3.31 VersionInfo

VersionInfo is a 7-byte value consisting of:

- 1-byte Major Application version

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

4.3.32 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.

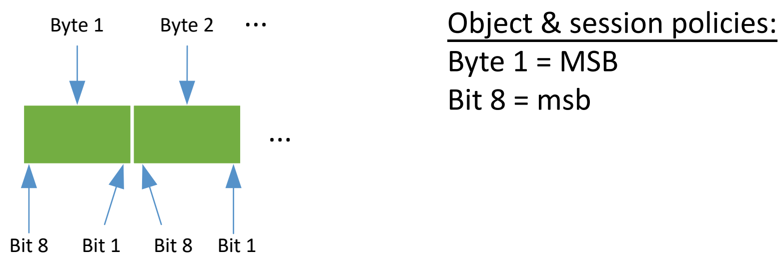

Figure 15. Policy notation

4.3.32.1 Session policy

The session policy header is coded as follows:

Table 50. 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.	0x8000	Y	2
POLICY_SESSION_MAX_TIMEOUT	Defines the time (in seconds) that a session remains opened. When the timeout expires, the session is closed.	0x4000	Y	2
POLICY_SESSION_ALLOW_REFRESH	Defines whether this session can be refreshed without losing context.	0x2000	N	
RFU	Other values reserved for future use	0x1FFF	n/a	

Setting a session policy is optional. If not set, there is no maximum number of APDU allowed, neither a session timeout. The session cannot be refreshed by default. In short, the default session policy is coded as: ‘02 0000’

4.3.32.2 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 51. Access rules

Access rule	Description	Bit in AR Header	Extension required?	Extension length
POLICY_OBJ_ALLOW_TLS_KDF	Allow TLS KDF	0x80000000	N	
POLICY_OBJ_ALLOW_TLS_PMS	Allow TLS pre master secret calculation	0x40000000	N	
POLICY_OBJ_FORBID_ALL	Explicitly forbid all operations	0x20000000	N	
POLICY_OBJ_ALLOW_SIGN	Allow signature or MAC generation	0x10000000	N	
POLICY_OBJ_ALLOW_VERIFY	Allow signature or MAC verification	0x08000000	N	
POLICY_OBJ_ALLOW_KA	Allow key agreement	0x04000000	N	
POLICY_OBJ_ALLOW_ENC	Allow encryption	0x02000000	N	
POLICY_OBJ_ALLOW_DEC	Allow decryption	0x01000000	N	
POLICY_OBJ_ALLOW_HKDF	Allow HKDF	0x00800000	N	
POLICY_OBJ_ALLOW_RFC3394_UNWRAP	Allow key wrapping (master key)	0x00400000	N	
POLICY_OBJ_ALLOW_READ	Allow to read the object	0x00200000	N	
POLICY_OBJ_ALLOW_WRITE	Allow to write the object	0x00100000	N	
POLICY_OBJ_ALLOW_GEN	Allow to (re)generate the object (only internally)	0x00080000	N	
POLICY_OBJ_ALLOW_DELETE	Allow to delete the object	0x00040000	N	
POLICY_OBJ_REQUIRE_SM	Require SCP03 or ECKey session secure messaging where secure messaging requires C_MAC and C_DECRYPTION set.	0x00020000	N	
POLICY_OBJ_REQUIRE_PCR_VALUE	Indicates that access to the object is allowed only if the given PCR object contains a certain value	0x00010000	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)	0x00008000	N	
POLICY_OBJ_ALLOW_IMPORT_EXPORT	Indicates that this object can be imported or exported	0x00001000	N	
POLICY_OBJ_FORBID_DERIVED_OUTPUT	Indicates if the object allows to output derived data	0x00000800	N	

Table 51. Access rules...*continued*

Access rule	Description	Bit in AR Header	Extension required?	Extension length
POLICY_OBJ_ALLOW_KDF_EXT_RANDOM	Indicates that client randoms can be inserted as argument for TLSPerformPRF.	0x00000400	N	
POLICY_OBJ_ALLOW_DERIVED_INPUT	Indicates that a key object uses derived output as key value.	0x00000100	Y	4 bytes master key for derivation.
POLICY_OBJ_INTERNAL_IV	Enforce internal IV generation	0x00000020	N	
RFU	Other values reserved for future use	0x0000003F	n/a	

4.4 Session management

See Sessions for general information on sessions.

4.4.1 Generic session commands

4.4.1.1 CreateSession

Creates a session on A5000.

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

Table 52. 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 53. CreateSession R-APDU Body

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

Table 54. CreateSession R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

Table 54. CreateSession R-APDU Trailer*...continued*

SW	Description
SW_CONDITIONS_NOT_SATISFIED	• The authenticator does not exist • The provided input data are incorrect. • The session is invalid.

4.4.1.2 ExchangeSessionData

Sets session policies for the current session.

Table 55. 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 56. ExchangeSessionData R-APDU Body

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

Table 57. ExchangeSessionData R-APDU Trailer

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

4.4.1.3 ProcessSessionCmd

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

If the command returns an error, the actual APDU command (in TLV[TAG_1]) is not executed.

Table 58. 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

Table 58. ProcessSessionCmd C-APDU*...continued*

Field	Value	Description
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 59. ProcessSessionCmd R-APDU Body

Value	Description
variable	as defined in the specific command section

Table 60. ProcessSessionCmd R-APDU Trailer

SW	Description
variable	as defined in the specific command section

4.4.1.4 RefreshSession

Refreshes a session on A5000, the policy of the running session can be updated; the rest of the session state remains.

Table 61. 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 62. RefreshSession R-APDU Body

Value	Description
-	

Table 63. RefreshSession R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.4.1.5 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 Application immediately needs to be reselected as further APDUs would not be handled successfully.

Table 64. CloseSession

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

Table 65. CloseSession R-APDU Body

Value	Description
None	

Table 66. CloseSession R-APDU Trailer

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

4.4.2 UserID session operations

4.4.2.1 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 67. VerifySessionUserID C-APDU

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

Table 67. VerifySessionUserID C-APDU*...continued*

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

Table 68. VerifySessionUserID R-APDU Body

Value	Description
-	

Table 69. VerifySessionUserID R-APDU Trailer

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

4.4.3 AESKey session operations

4.4.3.1 SCPInitializeUpdate

[SCP03] Section 7.1.1 shall be applied.

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

4.4.3.2 SCPExternalAuthenticate

[SCP03] Section 7.1.2 shall be applied.

4.4.4 ECKey session operations

4.4.4.1 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 70. ECKeySessionInternalAuthenticate C-APDU

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

Table 70. ECKeySessionInternalAuthenticate C-APDU*...continued*

Field	Value	Description
Lc	#(Payload)	
Payload	TLV[TAG_1]	Input data (see Table 71)
Le	0x00	

Table 71. ECKeySessionInternalAuthenticate C-APDU payload

TAG	SubTag	Length	Value
0xA6		Var	Control Reference Template
	0x4F	5-16	Application Instance AID
	0x90	3	SCP identifier and parameters: • SCP identifier must equal 0xAB • 2 byte parameters: 0x01 followed by a 1-byte GlobalPlatform security level
	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 72. ECKeySessionInternalAuthenticate R-APDU Body

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

Table 73. ECKeySessionInternalAuthenticate R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.5 Module management

4.5.1 SetLockState

Sets the Application 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 Application.

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 Application is locked.

The default TRANSIENT_LOCK state is LOCKED; there is no default PERSISTENT_LOCK state (depends on product configuration).

Table 74. 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 75. 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 76. SetLockState R-APDU Body

Value	Description
None	

Table 77. SetLockState R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.5.2 DisableObjectCreation

Disables object creation, either permanent or temporary and either only for new objects or also for existing objects.

Users need to decide if the switch is permanent or temporary:

- To permanently disable: LockIndicator = LOCK_PERSISTENT
- To temporary disable: LockIndicator = LOCK_TRANSIENT

Persistent locks remain over the lifetime of the product, transient locks are unlocked when deselecting the Application.

Users need to decide the level of restriction:

- To disable creation of new Secure Objects: RestrictMode = RESTRICT_NEW
- To disable creation of new Secure Objects and disable update of existing objects: RestrictMode = RESTRICT_ALL.

The following scenarios are applicable:

- When applying RESTRICT_ALL to LOCK_PERSISTENT, no object creation, modification or deletion is possible any more (permanently).
- When applying RESTRICT_NEW to LOCK_PERSISTENT, no new object creation or deletion is possible any more (permanently), but existing objects can still be modified.
- When applying RESTRICT_ALL to LOCK_TRANSIENT, no new object creation, modification or deletion is possible any more until Application deselect.
- When applying RESTRICT_NEW to LOCK_TRANSIENT, no new object creation or deletion is possible any more until Application deselect, but modification is possible except if RESTRICT_ALL is set on LOCK_PERSISTENT.

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

Table 78. DisableObjectCreation C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See Instruction
P1	P1_DEFAULT	See P1
P2	P2_RESTRICT	See P2
Lc	#(Payload)	
Payload	TLV[TAG_1]	1-byte LockIndicator
	TLV[TAG_2]	1-byte RestrictMode
Le		

Table 79. DisableObjectCreation R-APDU Body

Value	Description
None	

Table 80. DisableObjectCreation R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.5.3 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 Application APDU command will be refused by the Application 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 Application APDU command will be accepted by the Application.

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 81. 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 82. SetPlatformSCPRequest R-APDU Body

Value	Description
None	

Table 83. SetPlatformSCPRequest R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.5.4 SendCardManagerCommand

Sends a command to the Card Manager.

This function allows to send Card Manager commands from an external entity without the need to select the Card Manager explicitly, using the Application mechanisms to ensure a secure end-to-end channel for these commands to be communicated, e.g. using an ECKey session.

Note that the use of the command does not bypass any security mechanism from the Card Manager, i.e. users still must authenticate before performing a command that requires authentication.

Table 84. SendCardManagerCommand C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See Instruction
P1	P1_DEFAULT	See P1
P2	P2_CM_COMMAND	See P2
Lc	#(Payload)	Payload length
Payload	TLV[TAG_1]	APDU to be sent to the Card Manager.
Le	0x00	Expected response length

Table 85. SendCardManagerCommand R-APDU Body

Value	Description
TLV[TAG_1]	Byte array containing the Card Manager response.

Table 86. SendCardManagerCommand R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.5.5 TriggerSelfTest

Trigger a system health check for the system. When calling this command, a self-test is triggered in the operating system. When the test fails, the device might not respond with a R-APDU as the chip is reset.

Table 87. TriggerSelfTest C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See Instruction. In addition to INS_MGMT, users can set a flag to request an attested response.
P1	P1_DEFAULT	See P1
P2	P2_SANITY	See P2
Lc	#(Payload)	Payload length
Payload	TLV[TAG_1]	2-byte value from HealthCheckMode
	TLV[TAG_5]	4-byte attestation object identifier. Minimum policy: POLICY_OBJ_ALLOW_ATTESTATION *[Optional]* *[Conditional: only when attestation is requested.]*
	TLV[TAG_6]	1-byte AttestationAlgo *[Optional]* *[Conditional: only when attestation is requested.]*

Table 87. TriggerSelfTest C-APDU*...continued*

Field	Value	Description
	TLV[TAG_7]	16-byte freshness random *[Optional]* *[Conditional: only when attestation is requested.]*
Le	0x00	2-byte response + attested data (if an attestation flag is set).

Table 88. TriggerSelfTest R-APDU Body

Value	Description
TLV[TAG_1]	TLV containing 1-byte Result.
TLV[TAG_2]	TLV containing 18-byte chip unique ID *[Conditional: only when attestation is requested.]*
TLV[TAG_4]	TLV containing 0x0000. *[Conditional: only when attestation is requested.]*
TLV[TAG_TS]	TLV containing 12-byte timestamp *[Conditional: only when attestation is requested.]*
TLV[TAG_ATT_SIG]	TLV containing signature over the hashed plain C-APDU concatenated with tag, length and value of TLV[TAG_1], TLV[TAG_2], TLV[TAG_4] and TLV[TAG_TS] as returned by the Application. *[Conditional: only when attestation is requested.]*

Table 89. TriggerSelfTest R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.5.6 ReadState

Read the LockState, RestrictMode and PlatformSCPRequest status of the device. This command will return the current state of the device, regardless of transient or persistent lock state.

This command can be sent without applying platform SCP -even if PlatformSCPRequest is SCP_REQUIRED- and will also return a valid response when the LockState is LOCKED.

Table 90. ReadState C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_READ	See Instruction.
P1	P1_DEFAULT	See P1
P2	P2_STATE	See P2
Le	0x07	3-byte response.

Table 91. ReadState R-APDU Body

Value	Description
TLV[TAG_1]	TLV containing 3-byte result: LockState, RestrictMode and PlatformSCPRequest. If RestrictMode equals 0x00, no restrictions are applied.

Table 92. ReadState R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.6 Secure Object management

4.6.1 WriteSecureObject

Creates or writes to a Secure Object to the A5000.

Table 93. 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		

Table 94. WriteSecureObject R-APDU Body

Value	Description
-	

Table 95. WriteSecureObject R-APDU Trailer

SW	Description
SW_NO_ERROR	The file is created or updated successfully.

Table 96. WriteSecureObject variants

APDU	Reference	Description
WriteECKey	WriteECKey	Write an EC key pair, private key or public key.
WriteSymmKey	WriteSymmKey	Write an AES, DES or HMAC key.
WriteBinary	WriteBinary	Write to a binary file.

Table 96. WriteSecureObject variants*...continued*

APDU	Reference	Description
WriteUserID	WriteUserID	Write a userID value.
WriteCounter	WriteCounter	Write or increment a monotonic counter.
WritePCR	WritePCR	Write a PCR value.
ImportObject	ImportObject	Import an encrypted serialized Secure Object (previously exported)
ImportExternalObject	ImportExternal Object	Import an encrypted serialized Secure Object (externally created)

4.6.1.1 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 A5000.

If the object already exists, P1KeyType is ignored.

Warning: writing transient ECKey Secure Objects causes NVM write accesses.

Table 97. 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; else an error is returned and the object's value is not updated.]*
	TLV[TAG_POLICY_CHECK]	Byte array containing the object policy.to be compared against. *[Optional: if present, the existing policy must match this policy for the command to be executed.]* *[Conditional: only enforced when the key is passed as input, not checked for key generation]*
	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 Minimum policy:POLICY_OBJ_ALLOW_WRITE if the key allows to be updated by providing external data or POLICY_OBJ_ALLOW_GEN if the key allows to be regenerated on-chip.

Table 97. WriteECKey C-APDU...*continued*

Field	Value	Description
	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]*
	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]*
	TLV[TAG_11]	4-byte version, maximum is 134217727 (or 0x7FFFFFFF).. *[Optional]*

4.6.1.2 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 98. 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, else an error is returned and the object's value is not updated.]*
	TLV[TAG_POLICY_CHECK]	Byte array containing the object policy.to be compared against. *[Optional: if present, the existing policy must match this policy for the command to be executed.]*
	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 Minimum policy:POLICY_OBJ_ALLOW_WRITE if the key allows to be updated.

Table 98. WriteSymmKey C-APDU*...continued*

Field	Value	Description
	TLV[TAG_2]	4-byte KEK identifier, must be an AESKey identifier. *[Conditional: only when the key value is RFC3394 wrapped]*
	TLV[TAG_3]	Key value, either plain or RFC3394 wrapped. Minimum policy:POLICY_OBJ_ALLOW_RFC3394_UNWRAP
	TLV[TAG_4]	2-byte minimum tag length for AEAD operations, minimum is 4 and maximum is 16. *[Optional: default value = 16 bytes]* *[Conditional: only allowed for P1 = P1_AES]*
	TLV[TAG_5]	2-byte minimum output length for HKDF or TLS premaster secret calculation.. *[Conditional: only allowed for P1 = P1_HMAC]* *[Optional: Default value = 16 bytes for HMACKey; set to 0 and unused for other SymmKeys]*
	TLV[TAG_11]	4-byte version, maximum is 134217727 (or 0x7FFFFFFF). *[Optional: default value = 0 (= no versioning)]*

4.6.1.3 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 99. 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, else an error is returned and the object's value is not updated.]*
	TLV[TAG_POLICY_CHECK]	Byte array containing the object policy.to be compared against. *[Optional: if present, the existing policy must match this policy for the command to be executed.]*
	TLV[TAG_1]	4-byte object identifier Minimum policy:POLICY_OBJ_ALLOW_WRITE if the file allows to be updated. The policy will be applied immediately after the first C-APDU.

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

Field	Value	Description
	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 and the data will be initialized to zeroes; mandatory when the object exists]*
	TLV[TAG_11]	4-byte version, maximum is 134217727 (or 0x7FFFFFFF). *[Optional]*

4.6.1.4 WriteUserID

Creates a UserID object, setting the user identifier value.

UserIDs must be created as Authentication Object, userIDs as non-Authentication Objects are not supported.

Table 100. 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, else an error is returned and the object's value is not updated.]*
	TLV[TAG_MAX_ATTEMPTS]	2-byte maximum number of attempts. If 0 is given, this means unlimited. The maximum number of attempts must be smaller than 256. *[Optional: default = 0]* *[Conditional: only when the object identifier is not in use yet]*
	TLV[TAG_1]	4-byte object identifier.
	TLV[TAG_2]	Byte array containing 4 to 16 bytes user ID value.

4.6.1.5 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 101. 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, else an error is returned and the object's value is not updated.]*
	TLV[TAG_POLICY_CHECK]	Byte array containing the object policy.to be compared against. *[Optional: if present, the existing policy must match this policy for the command to be executed.]*
	TLV[TAG_1]	4-byte counter identifier. Minimum policy:POLICY_OBJ_ALLOW_WRITE if the counter allows to be updated.
	TLV[TAG_2]	2-byte counter size (1 up to 8 bytes). *[Conditional: only if object doesn't exist yet and TAG_3 is not given]*
	TLV[TAG_3]	Counter value *[Optional: - if object doesn't exist: must be present if TAG_2 is not given. - if object exists: if not present, increment by 1. if present, set counter to value.]*

4.6.1.6 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 the hash of its initial value (i.e., the hash of the value given 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 102. 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, else an error is returned and the object's value is not updated.]*

Table 102. WritePCR C-APDU*...continued*

Field	Value	Description
	TLV[TAG_POLICY_CHECK]	Byte array containing the object policy.to be compared against. *[Optional: if present, the existing policy must match this policy for the command to be executed.]*
	TLV[TAG_1]	4-byte PCR identifier. Minimum policy:POLICY_OBJ_ALLOW_WRITE if the PCR allows to be extended Optional policy:POLICY_OBJ_ALLOW_DELETE if the PCR allows to be reset to its initial value (next to regular Secure Object deletion).
	TLV[TAG_2]	Initial 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]*

4.6.1.7 ImportObject

Writes a serialized Secure Object to the A5000 (i.e., “import”). See SecureObjectImportExport for details on the import/export mechanism.

Table 103. ImportObject C-APDU

Field	Value	Description
P1	P1_DEFAULT	See P1
P2	P2_IMPORT	See P2
Payload	TLV[TAG_1]	4-byte identifier.
	TLV[TAG_3]	Serialized object (encrypted).

4.6.2 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. Only persistent Secure Objects can be created using this C-APDU, transient Secure Objects cannot be created using ImportExternalObject.

Table 104. ImportExternalObject C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_IMPORT_EXTERNAL	See Instruction

Table 104. ImportExternalObject C-APDU*...continued*

Field	Value	Description
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 105. ImportExternalObject R-APDU Body

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

Table 106. ImportExternalObject R-APDU Trailer

SW	Description
SW_NO_ERROR	The importExternalObject has finished succesfully.

4.6.3 ReadSecureObject

4.6.3.1 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, except if attestation is requested. In that case the object attributes will be returned, but not the key value.
- 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.

When attestation is requested, the secure object is read with attestation.

When the response length would exceed 256 bytes, the ReadObject command must be send as extended length APDU in order for the R-APDU to be in extended length format as well, else the command would return SW_CONDITIONS_NOT_SATISFIED.

Table 107. ReadObject C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_READ	See Instruction, in addition to INS_READ, users can set a flag to request reading with attestation.
P1	P1_DEFAULT	See P1
P2	P2_DEFAULT	See P2
Lc	#(Payload)	Payload Length.
	TLV[TAG_1]	4-byte object identifier Minimum policy: POLICY_OBJ_ALLOW_READ
	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_5]	4-byte attestation object identifier. Minimum policy: POLICY_OBJ_ALLOW_ATTESTATION *[Optional]* *[Conditional: only when attestation is requested]*
	TLV[TAG_6]	1-byte AttestationAlgo *[Optional]* *[Conditional: only when attestation is requested]*

Table 107. ReadObject C-APDU...*continued*

Field	Value	Description
	TLV[TAG_7]	16-byte freshness random *[Optional]* *[Conditional: only when attestation is requested]*
Le	0x00	

Table 108. ReadObject R-APDU Body

Value	Description
TLV[TAG_1]	Data read from the Secure Object.
TLV[TAG_2]	18-byte Chip unique ID. *[Conditional: only when attestation is requested.]*
TLV[TAG_3]	Byte array containing the Secure Object attributes. *[Conditional: only when attestation is requested.]*
TLV[TAG_4]	2-byte Secure Object size.
TLV[TAG_TS]	12-byte timestamp. *[Conditional: only when attestation is requested.]*
TLV[TAG_ATT_SIG]	Signature applied over the hashed plain C-APDU concatenated with tag, length and value of TLV[TAG_1], TLV[TAG_2], TLV[TAG_3], TLV[TAG_4] and TLV[TAG_TS] as returned by the Application. *[Conditional: only when attestation is requested.]*

Table 109. ReadObject R-APDU Trailer

SW	Description
SW_NO_ERROR	The value is read successfully.
SW_CONDITIONS_NOT_SATISFIED	The value cannot be read.

4.6.3.2 ReadAttributes

Reads the Object Attributes of a Secure Object (without the value of the Secure Object).

The response will contain a TLV[TAG_3] containing the object attributes.

When attestation is requested by putting an attestation flag into the INS byte, the secure object is read with attestation.

Table 110. ReadAttributes C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_READ	See Instruction, in addition to INS_READ, users can set a flag to request reading with attestation.
P1	P1_DEFAULT	See P1

Table 110. ReadAttributes C-APDU...*continued*

Field	Value	Description
P2	P2_ATTRIBUTES	See P2
Lc	#(Payload)	Payload Length.
	TLV[TAG_1]	4-byte object identifier Minimum policy: POLICY_OBJ_ALLOW_READ
	TLV[TAG_5]	4-byte attestation object identifier. *[Optional]* *[Conditional: only when attestation is requestedt]*
	TLV[TAG_6]	1-byte AttestationAlgo *[Optional]* *[Conditional: only when attestation is requestedt]*
	TLV[TAG_7]	16-byte freshness random *[Optional]* *[Conditional: only when attestation is requestedt]*
Le	0x00	

Table 111. ReadAttributes R-APDU Body

Value	Description
TLV[TAG_2]	18-byte Chip unique ID. *[Conditional: only when attestation is requested.]*
TLV[TAG_3]	Byte array containing the Secure Object attributes.
TLV[TAG_4]	2-byte Secure Object size. *[Conditional: only when attestation is requested.]*
TLV[TAG_TS]	12-byte timestamp. *[Conditional: only when attestation is requested.]*
TLV[TAG_ATT_SIG]	Signature applied over the hashed plain C-APDU concatenated with tag, length and value of TLV[TAG_2], TLV[TAG_3], TLV[TAG_4] and TLV[TAG_TS] as returned by the Application. *[Conditional: only when attestation is requested.]*

Table 112. ReadAttributes R-APDU Trailer

SW	Description
SW_NO_ERROR	The read is done successfully.

4.6.3.3 ExportObject

Reads a transient Secure Object from A5000. See SecureObjectImportExport for details on the import/export mechanism.

Table 113. 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
Le	0x00	

Table 114. ExportObject R-APDU Body

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

Table 115. ExportObject R-APDU Trailer

SW	Description
SW_NO_ERROR	The file is created or updated successfully.

4.6.4 ManageSecureObject

4.6.4.1 ReadType

Get the type of a Secure Object.

Table 116. ReadType C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_READ	See Instruction
P1	P1_DEFAULT	See P1
P2	P2_TYPE	See P2
Lc	#(Payload)	
	TLV[TAG_1]	4-byte object identifier. Minimum policy: POLICY_OBJ_ALLOW_READ
Le	0x00	

Table 117. ReadType R-APDU Body

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

Table 118. ReadType R-APDU Trailer

SW	Description
SW_NO_ERROR	Data is returned successfully.

4.6.4.2 ReadSize

Get the Secure Object size for the specified Secure Object.

Table 119. 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. Minimum policy: POLICY_OBJ_ALLOW_READ
Le	0x00	

Table 120. ReadSize R-APDU Body

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

Table 121. ReadSize R-APDU Trailer

SW	Description
SW_NO_ERROR	Data are returned successfully.
SW_ CONDITIONS_ NOT_SATISFIED	Data are not returned.

4.6.4.3 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 Application 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 122. 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 123. ReadIDList R-APDU Body

Value	Description
TLV[TAG_1]	1-byte MoreIndicator
TLV[TAG_2]	Byte array containing 4-byte identifiers.

Table 124. ReadIDList R-APDU Trailer

SW	Description
SW_NO_ERROR	Data is returned successfully.

4.6.4.4 CheckObjectExists

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

Table 125. 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 126. CheckObjectExists R-APDU Body

Value	Description
TLV[TAG_1]	1-byte Result

Table 127. CheckObjectExists R-APDU Trailer

SW	Description
SW_NO_ERROR	Data is returned successfully.

4.6.4.5 DeleteSecureObject

Triggers the deletion of a Secure Object. Garbage collection is triggered.

If the object origin = ORIGIN_PROVISIONED, an error will be returned and the object is not deleted, even if the policy allows deletion.

Table 128. DeleteSecureObject C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See Instruction
P1	P1_DEFAULT	See P1
P2	P2_DELETE_OBJECT	See P2
Lc	#(Payload)	
	TLV[TAG_1]	4-byte existing Secure Object identifier. Minimum policy:POLICY_OBJ_ALLOW_DELETE
Le	-	

Table 129. DeleteSecureObject R-APDU Body

Value	Description
-	

Table 130. DeleteSecureObject R-APDU Trailer

SW	Description
SW_NO_ERROR	The file is created or updated successfully.

4.7 EC curve management

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

4.7.1 CreateECCurve

Create an EC curve listed in ECCurve.

This function must be called for all supported curves in ECCurve when the curve is to be used.

If the curve is already fully initilized, SW_CONDTIONS_NOT_SATISFIED will be returned; users have to call DeleteECCurve if the curve needs to be recreated.

Table 131. 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 132. CreateECCurve R-APDU Body

Value	Description
-	

Table 133. CreateECCurve R-APDU Trailer

SW	Description
SW_NO_ERROR	Data is returned successfully.

4.7.2 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.

If the curve is already fully initilized, SW_CONDTIONS_NOT_SATISFIED will be returned; users have to call DeleteECCurve if the parameters need to be reset.

Users have to set all 5 curve parameters for the curve to be usable. Once all curve parameters are given, the secure authenticator will check if all parameters are correct and return SW_NO_ERROR. If the values of the parameters do not match the expected curve parameters, an error will be returned.

This function must be called for all supported curves in ECCurve when the curve is to be used.

Table 134. 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)	

Table 134. SetECCurveParam C-APDU...*continued*

Field	Value	Description
	TLV[TAG_1]	1-byte curve identifier, from ECCurve
	TLV[TAG_2]	1-byte ECCurveParam
	TLV[TAG_3]	Bytestring containing curve parameter value.

Table 135. SetECCurveParam R-APDU Body

Value	Description
-	

Table 136. SetECCurveParam R-APDU Trailer

SW	Description
SW_NO_ERROR	Data is returned successfully.

4.7.3 GetECCurveID

Get the curve associated with an EC key.

Table 137. GetECCurveID C-APDU

Field	Value	Description
CLA	0x80	
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 138. GetECCurveID R-APDU Body

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

Table 139. GetECCurveID R-APDU Trailer

SW	Description
SW_NO_ERROR	Data is returned successfully.

4.7.4 ReadECCurveList

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

Table 140. 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 141. 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 142. ReadECCurveList R-APDU Trailer

SW	Description
SW_NO_ERROR	Data is returned successfully.

4.7.5 DeleteECCurve

Deletes an EC curve. Garbage collection is triggered.

Table 143. 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 144. DeleteECCurve R-APDU Body

Value	Description
-	

Table 145. DeleteECCurve R-APDU Trailer

SW	Description
SW_NO_ERROR	Data is returned successfully.

4.8 Crypto Object management

4.8.1 CreateCryptoObject

Creates a Crypto Object on the A5000. Once the Crypto Object is created, it is bound to the user who created the Crypto Object, no other user can use the Crypto Object.

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

Table 146. 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, MACAlgo (depending on TAG_2) or AEADMode.

Table 147. CreateCryptoObject R-APDU Body

Value	Description
-	

Table 148. CreateCryptoObject R-APDU Trailer

SW	Description
SW_NO_ERROR	The Crypto Object is created successfully.

4.8.2 ReadCryptoObjectList

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

Table 149. 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 150. 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 151. ReadCryptoObjectList R-APDU Trailer

SW	Description
SW_NO_ERROR	Data is returned successfully.

4.8.3 DeleteCryptoObject

Deletes a Crypto Object on the A5000. Garbage collection is triggered.

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 152. DeleteCryptoObject C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_MGMT	See Instruction
P1	P1_CRYPTO_OBJ	See P1
P2	P2_DELETE_OBJECT	See P2
Lc	#(Payload)	Payload length
Payload	TLV[TAG_1]	2-byte Crypto Object identifier

Table 153. DeleteCryptoObject R-APDU Body

Value	Description
-	

Table 154. DeleteCryptoObject R-APDU Trailer

SW	Description
SW_NO_ERROR	The file is created or updated successfully.

4.9 Crypto operations EC

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

4.9.1 Signature generation

4.9.1.1 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 155. 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. Minimum policy:POLICY_OBJ_ALLOW_SIGN Optional policy: POLICY_OBJ_FORBID_DERIVED_OUPUT to prevent output to host.
	TLV[TAG_2]	1-byte ECSignatureAlgo.
	TLV[TAG_3]	Byte array containing hashed input data; the hash algorithm must match the ECSignatureAlgo in TLV[TAG_2].
Le	0x00	Expecting ASN.1 signature

Table 156. ECDSASign R-APDU Body

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

Table 157. ECDSASign R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.9.2 Signature verification

4.9.2.1 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 158. 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. Minimum policy:POLICY_OBJ_ALLOW_VERIFY
	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 159. ECDSAVerify R-APDU Body

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

Table 160. ECDSAVerify R-APDU Trailer

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

4.9.3 Shared secret generation

4.9.3.1 ECDHGenerateSharedSecret

The ECDHGenerateSharedSecret command computes a shared secret using an EC private key on A5000 and an external public key provided by the caller. The external public key can either be passed as byte array (using TLV[TAG_2]) or via a Secure Object identifier to an ECPublicKey object (using TLV[TAG_3]).

The output shared secret is returned to the caller (if TLV[TAG_7] is not used) or stored inside an AESKey or HMACKey (using TLV[TAG_7]).

Using P2 equal to P2_DH will return or store output in big endian format. Using P2 equal to P2_DH_REVERSE will return or store output in little endian format. Note that -when storing the public key into a Secure Object- the byte order must also be reversed for correct shared secret generation.

Table 161. ECDHGenerateSharedSecret C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	Instruction
P1	P1_EC	See P1
P2	P2_DH or P2_DH_REVERSE	See P2
Lc	#(Payload)	
Payload	TLV[TAG_1]	4-byte identifier of the key pair or private key. Minimum policy: POLICY_OBJ_ALLOW_KA Optional policy: POLICY_OBJ_FORBID_DERIVED_OUPUT to prevent output to host (i.e. mandate use of TLV[TAG_7]).
	TLV[TAG_2]	Byte array containing external public key (see ECKey). *[Conditional: only when TAG_3 is absent]*
	TLV[TAG_3]	4-byte identifier of the external EC public key. Minimal policy: POLICY_OBJ_ALLOW_KA *[Conditional: only when TAG_2 is absent]*
	TLV[TAG_4]	1-byte ECDHAlgo *[Optional: default is EC_SVDP_DH_PLAIN]*
	TLV[TAG_7]	4-byte identifier of the target Secure Object, either of type AESKey or HMACKey. Minimum policy: POLICY_OBJ_ALLOW_WRITE or POLICY_OBJ_ALLOW_DERIVED_INPUT with the 4-byte identifier of TLV[TAG_1] as extension to restrict key derivation. *[Optional]*
Le	0x00	Expected shared secret length.

Table 162. ECDHGenerateSharedSecret R-APDU Body

Value	Description
TLV[TAG_1]	The returned shared secret. *[Conditional: only when the input does not contain TLV[TAG_7].}*

Table 163. ECDHGenerateSharedSecret R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.9.4 EC Point Multiplication

4.9.4.1 ECPointMultiply

The ECPointMultiply command computes an ECC point on the curve using an EC private key on A5000 and an external public key provided by the caller. The external public key can either be passed as byte array (using TLV[TAG_2]) or via a Secure Object identifier to an ECPublicKey object (using TLV[TAG_3]).

The output is returned to the caller (if TLV[TAG_7] is not used) or stored inside an ECPublicKey object (using TLV[TAG_7]).

Table 164. ECPointMultiply C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	Instruction
P1	P1_EC	See P1
P2	P2_ECPM	See P2
Lc	#(Payload)	
Payload	TLV[TAG_1]	4-byte identifier of the key pair or private key. Minimum policy: POLICY_OBJ_ALLOW_KA Optional policy: POLICY_OBJ_FORBID_DERIVED_OUPUT would prevent to execute this command succesfully.
	TLV[TAG_2]	Byte array containing external public key (see ECKey). *[Conditional: only when TAG_3 is absent]*
	TLV[TAG_3]	4-byte identifier of the external EC public key. Minimal policy: POLICY_OBJ_ALLOW_KA *[Conditional: only when TAG_2 is absent]*
	TLV[TAG_4]	1-byte ECPMAlgo
	TLV[TAG_7]	4-byte identifier of the target Secure Object; this needs to be an ECPublicKey of the expected length to store the result. Minimum policy: POLICY_OBJ_ALLOW_WRITE or POLICY_OBJ_ALLOW_DERIVED_INPUT with the 4-byte identifier of TLV[TAG_1] as extension to restrict key derivation. *[Optional]*
Le	0x00	Expected EC point length.

Table 165. ECPointMultiply R-APDU Body

Value	Description
TLV[TAG_1]	The returned EC point. Format depends on the ECPMAlgo. *[Conditional: only when the input does not contain TLV[TAG_7].}*

Table 166. ECPointMultiply R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.10 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).

4.10.1 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 167. 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)	

Table 167. CipherInit C-APDU*...continued*

Field	Value	Description
Payload	TLV[TAG_1]	4-byte identifier of the key object. Minimum policy:POLICY_OBJ_ALLOW_ENC or POLICY_OBJ_ALLOW_DEC depending on P2. Optional policy: POLICY_OBJ_FORBID_EXTERNAL_IV Optional policy: POLICY_OBJ_FORBID_DERIVED_OUPUT would prevent to execute this command succesfully.
	TLV[TAG_2]	2-byte Crypto Object identifier
	TLV[TAG_4]	Byte array containing the initialization vector for AES [16 bytes] for DES [8 bytes] or a 2-byte value containing the length of the initialization vector to be generated (only when P2 = P2_ENCRYPT and the CyptoObject type equals CC_CIPHER with subtype equal to AES_CTR.. *[Optional]* *[Conditional: only when the Crypto Object type equals CC_CIPHER, subtype is not including ECB]*
Le	-	

Table 168. CipherInit R-APDU Body

Value	Description
TLV[TAG_3]	Byte array containing the initialization vector. *[Conditional: only when P2 equals P2_ENCRYPT_ONESHOT and TLV[TAG_4] in the C-APDU contains 2 bytes Value.]*

Table 169. CipherInit R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.10.2 CipherUpdate

Update a cipher context.

Table 170. CipherUpdate C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	Instruction
P1	P1_CIPHER	See P1
P2	P2_UPDATE	See P2
Lc	#(Payload)	

Table 170. CipherUpdate C-APDU*...continued*

Field	Value	Description
Payload	TLV[TAG_2]	2-byte Crypto Object identifier
	TLV[TAG_3]	Byte array containing input data
Le	0x00	Expecting returned data.

Table 171. CipherUpdate R-APDU Body

Value	Description
TLV[TAG_1]	Output data

Table 172. CipherUpdate R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.10.3 CipherFinal

Finish a sequence of cipher operations.

Table 173. 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 174. CipherFinal R-APDU Body

Value	Description
TLV[TAG_1]	Output data

Table 175. CipherFinal R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.10.4 CipherOneShot

Encrypt or decrypt data in one shot mode.

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

Table 176. 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. Minimum policy:POLICY_OBJ_ALLOW_ENC or POLICY_OBJ_ALLOW_DEC depending on P2. Optional policy: POLICY_OBJ_FORBID_EXTERNAL_IV Optional policy: POLICY_OBJ_FORBID_DERIVED_OUPUT would prevent to execute this command succesfully.
	TLV[TAG_2]	1-byte CipherMode
	TLV[TAG_3]	Byte array containing input data.
	TLV[TAG_4]	Byte array containing the initialization vector for AES [16 bytes] for DES [8 bytes] (if more bytes are passed for DES they are ignored) or a 2-byte value containing the length of the initialization vector to be generated (only when P2 = P2_ENCRYPT_ONESHOT and the CipherMode equals AES_CTR). *[Optional]* *[Conditional: when the CipherMode requires an initialization vector, this is a mandatory input]*
Le	0x00	Expecting return data.

Table 177. CipherOneShot R-APDU Body

Value	Description
TLV[TAG_1]	Output data
TLV[TAG_3]	Byte array containing the initialization vector . *[Conditional: only when P2 equals P2_ENCRYPT_ONESHOT TLV[TAG_4] in the C-APDU contains 2 bytes Value.]*

Table 178. CipherOneShot R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.11 Authenticated Encryption with Associated Data (AEAD)

AEAD operations can be done using a Secure Object of type AESKey.

AEADMode indicates the algorithm to be applied.

There are 2 options to use AEAD crypto modes:

- in one shot mode – 1 call to process data
- in multi shot mode – multiple calls to process data (init/update/final sequence).

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 AEAD operation in multiple steps involves NVM write access.

Notes on using AEAD crypto operations:

- AEADMode equal to AES_GCM supports IV lengths of 12 up to 60 bytes. Any other input will return an error.
- AEADMode equal to AES_GCM can be used for GMAC operations by omitting the data input and only send Additional Authenticated Data (AAD) input.
- AEADMode equal to AES_CCM supports tag lengths of 4, 6, 8, 10, 12, 14 and 16 bytes only. Any other input will return an error.
- AEADMode equal to AES_CCM supports nonce length of 7,8,9, 10, 11, 12 or 13 bytes only. Any other input will return an error.
- AEADMode equal to AES_CCM is only available in multi shot mode.
- It is up to the user to send AAD and (plain or encrypted) input data 16-byte aligned, both for AAD and data to encrypt or decrypt (except for the last block of the AAD and the last block of the data to encrypt or decrypt). AAD must always be sent before (plain or encrypted) input data. For AEADOneShot, these can be passed together as input.

4.11.1 AEADInit

Initialize an authentication encryption or decryption with associated data. The Crypto Object keeps the state of the AEAD operation until it’s finalized or deleted. Once the AEADFinal function is executed successfully, the Crypto Object state returns to the state immediately after the previous AEADInit function.

When TLV[TAG_5] contains a 2-byte Value, the initialization vector will be randomly generated (matching the requested length) and will be returned in the response command; else the TLV[TAG_5] must contain the IV to be used.

Table 179. AEADInit C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	Instruction
P1	P1_AEAD	See P1
P2	P2_ENCRYPT or P2_ DECRYPT	See P2
Lc	#(Payload)	

Table 179. AEADInit C-APDU*...continued*

Field	Value	Description
Payload	TLV[TAG_1]	4-byte identifier of the AESKey Secure object. Minimum policy:POLICY_OBJ_ALLOW_ENC or POLICY_OBJ_ALLOW_DEC depending on P2. Optional policy: POLICY_OBJ_FORBID_EXTERNAL_IV Optional policy: POLICY_OBJ_FORBID_DERIVED_OUPUT would prevent to execute this command succesfully.
	TLV[TAG_2]	2-byte Crypto Object identifier
	TLV[TAG_5]	Byte array containing the initialization vector [12 bytes until 60 bytes] or a 2-byte value containing the initialization vector length when P2 equals P2_ENCRYPT and the CyptoObject type equals CC_CIPHER with subtype equal to AES_GCM or AES_CCM.
	TLV[TAG_6]	Byte array containing 2-byte AAD length. *[Conditional: needed if AEADMode equals AES_CCM]*
	TLV[TAG_7]	Byte array containing 2-byte message length. *[Conditional: needed if AEADMode equals AES_CCM]*
	TLV[TAG_8]	Byte array containing 2-byte mac size. This must be equal to or higher than the minimum tag length attribute of the key identified in TLV[TAG_1]. *[Conditional: needed if AEADMode equals AES_CCM].*
Le	-	

Table 180. AEADInit R-APDU Body

Value	Description
TLV[TAG_3]	Byte array containing the used initialization vector. It remains valid until deselect, AEADInit, AEADFinal or AEADOneShot is called. *[Conditional: Only when P2 equals P2_ ENCRYPT and TLV[TAG_5] in the C-APDU contains 2 bytes Value.]*

Table 181. AEADInit R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.11.2 AEADUpdate

Update a Crypto Object of type CC_AEAD.

The user either needs to send input data or Additional Authenticated Data (AAD), but not both at once.

Note that the R-APDU does not always contain output data, even if input data are passed to the C-APDU. These might only be returned when calling AEADFinal.

Table 182. AEADUpdate C-APDU

Field	Value	Description
CLA	0x80	

Table 182. AEADUpdate C-APDU*...continued*

Field	Value	Description
INS	INS_CRYPTO	Instruction
P1	P1_AEAD	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 *[Conditional: only when TLV[TAG_4] is not present]* *[Optional]*
	TLV[TAG_4]	Byte array containing Additional Authenticated Data. *[Conditional: only when TLV[TAG_3] is not present]* *[Optional]*
Le	0x00	Expecting returned data.

Table 183. AEADUpdate R-APDU Body

Value	Description
TLV[TAG_1]	Output data *[Conditional: only when output data is available]*

Table 184. AEADUpdate R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.11.3 AEADFinal

Finish a sequence of AEAD operations. The AEADFinal command provides the computed GMAC or indicates whether the GMAC is correct depending on the P2 parameters passed during AEADInit. The length of the GMAC is always 16 bytes when P2 equals P2_ENCRYPT. When P2 equals P2_DECRYPT, the minimum tag length to pass is 4 bytes.

Table 185. AEADFinal C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	Instruction
P1	P1_AEAD	See P1
P2	P2_FINAL	See P2
Lc	#(Payload)	
Payload	TLV[TAG_2]	2-byte Crypto Object identifier

Table 185. AEADFinal C-APDU*...continued*

Field	Value	Description
	TLV[TAG_6]	Byte array containing tag to verify. The tag length must be equal to or higher than the minimum tag lengthattribute of the key identified in TLV[TAG_1] ot the AEADInit command. *[Conditional] When the mode is decrypt and verify (i.e. AEADInit has been called with P2 = P2_DECRYPT).*
Le	0x00	Expected returned data.

Table 186. AEADFinal R-APDU Body

Value	Description
TLV[TAG_1]	Output data *[Conditional: only when output data is available]*
TLV[TAG_2]	Byte array containing tag (if P2 = P2_ENCRYPT) or byte array containing Result (if P2 = P2_DECRYPT)

Table 187. AEADFinal R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.11.4 AEADOneShot

Authenticated encryption or decryption with associated data in one shot mode.

The key object must be an AES key.

When the AEADMode equals AES_GCM, the length of AAD + length of data should be limited to 888 bytes - the total C-APDU buffer length, where length of AAD and length of data are both rounded up to a multiple of 16, e.g. a C-APDU where data length = 397 bytes, AAD length = 20 bytes and IV length = 12 bytes is normally 456 bytes long (= 7 bytes extended C-APDU header + 6 bytes for TLV[TAG_1] + 3 bytes for TLV[TAG_2] + 401 bytes for TLV[TAG_3] + 22 bytes for TLV[TAG_4] + 14 bytes for bytes for TLV[TAG_5] + 3 bytes Le) would be fine as 888 - 456 >= (400 + 32).

When P2 equals P2_ENCRYPT_ONE_SHOT, the AEADOneShot command returns the encrypted data and computed authentication tag. When passed to the command, the authentication tag length must be at least the size that is defined during key creation (default 16 bytes). See Section 4.6.1.2 for details.

The length of the authentication tag is always 16 bytes when P2 equals P2_ENCRYPT_ONESHOT.

When P2 equals P2_DECRYPT_ONESHOT:

- the minimum authentication tag length to pass is defined during key creation (default 16 bytes). See Section 4.6.1.2 for details.

- when the authentication tag is not correct, only the result will be returned, no output data will be present.

Table 188. AEADOneShot C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	Instruction
P1	P1_AEAD	See P1
P2	P2_ENCRYPT_ ONESHOT or P2_DECRYPT_ ONESHOT	See P2
Lc	#(Payload)	
Payload	TLV[TAG_1]	4-byte identifier of the AESKey Secure object. Minimum policy:POLICY_OBJ_ALLOW_ENC or POLICY_OBJ_ALLOW_DEC depending on P2. Optional policy: POLICY_OBJ_FORBID_EXTERNAL_IV. Optional policy: POLICY_OBJ_FORBID_DERIVED_OUPUT would prevent to execute this command succesfully.
	TLV[TAG_2]	1-byte AEADMode except AEAD_CCM.
	TLV[TAG_3]	Byte array containing input data. *[Optional]*
	TLV[TAG_4]	Byte array containing Additional Authenticated Data. *[Optional]*
	TLV[TAG_5]	If AEADMode = AES_GCM: Byte array containing an initialization vector (IV length = 12 up to 60 bytes) or 2-byte value containing the requested initialization vector length.
	TLV[TAG_6]	2-byte value containing the requested tag length (if P2 equals P2_ENCRYPT_ONESHOT) or a 4 up to 16-byte array containing the authentication tag to verify (if P2 equals P2_DECRYPT_ONESHOT). The tag length must be equal to or higher than the minimum tag lengthattribute of the key identified in TLV[TAG_1].
Le	0x00	Expecting return data.

Table 189. AEADOneShot R-APDU Body

Value	Description
TLV[TAG_1]	Byte array containing output data.
TLV[TAG_2]	Byte array containing tag (if P2 = P2_ENCRYPT_ONESHOT) or byte array containing Result (if P2 = P2_DECRYPT_ONESHOT)
TLV[TAG_3]	Byte array containing the initialization vector *[Conditional: Only when P2 equals P2_ ENCRYPT_ONESHOT and TLV[TAG_5] in the C-APDU contains 2 bytes Value]*

Table 190. AEADOneShot R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.12 Message Authentication Codes

There are 2 options to use Message Authentication Codes on A5000:

- 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.

4.12.1 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 191. 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. Minimum policy:POLICY_OBJ_ALLOW_SIGN or POLICY_OBJ_ALLOW_VERIFY depending on P2. Optional policy: POLICY_OBJ_FORBID_DERIVED_OUPUT would prevent to execute this command succesfully (only when P2 equals P2_GENERATE).
	TLV[TAG_2]	2-byte Crypto Object identifier
Le	0x00	

Table 192. MACInit R-APDU Body

Value	Description
-	-

Table 193. MACInit R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.12.2 MACUpdate

Update a MAC operation.

Table 194. 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.
	TLV[TAG_2]	2-byte Crypto Object identifier
Le	-	

Table 195. MACUpdate R-APDU Body

Value	Description
-	

Table 196. MACUpdate R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.12.3 MACFinal

Finalize a MAC operation.

Table 197. 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

Table 197. MACFinal C-APDU...*continued*

Field	Value	Description
	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 198. 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 199. MACFinal R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.12.4 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 200. 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. Minimum policy:POLICY_OBJ_ALLOW_SIGN or POLICY_OBJ_ALLOW_VERIFY depending on P2. Optional policy: POLICY_OBJ_FORBID_DERIVED_OUPUT to prevent output to host (only when P2 equals P2_GENERATE_ONESHOT).
	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 201. MACOneShot R-APDU Body

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

Table 202. MACOneShot R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.13 Key Derivation Functions

4.13.1 HKDF

Perform HMAC Key Derivation Function according to [RFC5869]. There are 2 options:

- Perform the full algorithm, i.e. Extract-and-Expand => see HKDFExtractAndExpand
- Perform only the Expand step, i.e. skip Extract => see HKDFExpandOnly

The output of the HKDF functions can be either:

- sent back to the caller => precondition: none of the input Secure Objects -if present- shall have a policy POLICY_OBJ_FORBID_DERIVED_OUTPUT set.
- be stored in a Secure Object => precondition: the Secure Object must be created upfront and the size must exactly match the expected length.

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.

4.13.1.1 HKDFExtractAndExpand

The full HKDF algorithm is executed, i.e. Extract-And-Expand.

The caller must provide a salt length (0 up to 64 bytes). If salt length equals 0 or salt is not provided as input, the default salt will be used.

If the output is stored into an object, the object indicated in TLV[TAG_7] must be created before calling this function.

Table 203. HKDFExtractAndExpand C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	See Instruction
P1	P1_DEFAULT	See P1
P2	P2_HKDF	See P2
Lc	#(Payload)	
Payload	TLV[TAG_1]	4-byte HMACKey identifier (= IKM). Minimum policy: POLICY_OBJ_ALLOW_HKDF Optional policy: POLICY_OBJ_FORBID_DERIVED_OUPUT to prevent output to host.

Table 203. HKDFExtractAndExpand C-APDU*...continued*

Field	Value	Description
	TLV[TAG_2]	1-byte DigestMode (except DIGEST_NO_HASH and DIGEST_SHA224)
	TLV[TAG_3]	Byte array (0-64 bytes) containing salt. If EXTCFG_CRYPTO_HKDF_FORBID_IN_OUT_LT_112BIT is set, the minimum is 14 bytes. *[Optional]* *[Conditional: only when TLV[TAG_6] is absent.]*
	TLV[TAG_4]	Info: The context and information to apply (1 to 80 bytes). *[Optional]*
	TLV[TAG_5]	2-byte requested length (L): 1 up to 768 bytes. If EXTCFG_CRYPTO_HKDF_FORBID_IN_OUT_LT_112BIT is set, the minimum is 14 bytes. If a minimum output length is set on the key from TLV[TAG_1], the requested length must be equal or bigger than the minimum output length.
	TLV[TAG_6]	4-byte HMACKey identifier containing salt. Minimum policy: POLICY_OBJ_ALLOW_USAGE_AS_HMAC_PEPPER Optional policy: POLICY_OBJ_FORIBD_DERIVED_OUPUT to prevent output to host. *[Optional]* *[Conditional: only when TLV[TAG_3] is absent]*
	TLV[TAG_7]	4-byte identifier of the target Secure Object; this must be an HMACKey or AESKey. For HMACKey, minimum output length applies. Minimum policy:POLICY_OBJ_ALLOW_WRITE or POLICY_OBJ_ALLOW_DERIVED_INPUT with the 4-byte HMACKey identifier from TLV[TAG_1] as extension to restrict key derivation. *[Optional]*
Le	0x00	

Table 204. HKDFExtractAndExpand R-APDU Body

Value	Description
TLV[TAG_1]	HKDF output. *[Conditional: only when the input does not contain TLV[TAG-7]]*

Table 205. HKDFExtractAndExpand R-APDU Trailer

SW	Description
SW_NO_ERROR	The HKDF is executed successfully.

4.13.1.2 HKDFExpandOnly

Only step 2 of the algorithm is executed, i.e. Expand only.

Using an IV as input parameter results in a FIPS compliant NIST SP800-108 KDF in Feedback Mode where K[0] is the provided IV. This KDF is using a 8-bit counter, AFTER_FIXED counter location.

If the output is stored into an object, the object indicated in TLV[TAG_7] must be created before calling this function.

Table 206. HKDFExpandOnly C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	See Instruction
P1	P1_DEFAULT	See P1
P2	P2_HKDF_EXPAND_ ONLY	See P2
Lc	#(Payload)	
Payload	TLV[TAG_1]	4-byte HMACKey identifier (= PRK). Minimum policy: POLICY_OBJ_ALLOW_HKDF Optional policy: POLICY_OBJ_FORBID_DERIVED_OUPUT to prevent output to host.
	TLV[TAG_2]	1-byte DigestMode (except DIGEST_NO_HASH and DIGEST_SHA224)
	TLV[TAG_3]	Byte array (0-64 bytes) containing IV. *[Optional]* *[Conditional: only when TLV[TAG_6] is absent.]*
	TLV[TAG_4]	Info: The context and information to apply (1 to 80 bytes). *[Optional]*
	TLV[TAG_5]	2-byte requested length (L): 1 up to 768 bytes. If EXTCFG_CRYPTO_HKDF_FORBID_IN_OUT_LT_ 112BIT is set, the minimum is 14 bytes. If a minimum output length is set on the key from TLV[TAG_1], the requested length must be equal or bigger than the minimum output length.
	TLV[TAG_6]	4-byte HMACKey identifier containing IV. Minimum policy: POLICY_OBJ_ALLOW_USAGE_AS_HMAC_PEPPER Optional policy: POLICY_OBJ_FORIBD_DERIVED_OUPUT to prevent output to host. *[Optional]* *[Conditional: only when TLV[TAG_3] is absent]*
	TLV[TAG_7]	4-byte identifier of the target Secure Object; this must be an HMACKey or AESKey. For HMACKey, minimum output length applies. Minimum policy:POLICY_OBJ_ALLOW_WRITE or POLICY_OBJ_ALLOW_DERIVED_INPUT with the 4-byte HMACKey identifier from TLV[TAG_1] as extension to restrict key derivation. *[Optional]*

Table 206. HKDFExpandOnly C-APDU*...continued*

Field	Value	Description
Le	0x00	

Table 207. HKDFExpandOnly R-APDU Body

Value	Description
TLV[TAG_1]	HKDF output. *[Conditional: only when the input does not contain TLV[TAG-7]]*

Table 208. HKDFExpandOnly R-APDU Trailer

SW	Description
SW_NO_ERROR	The HKDF is executed successfully.

4.14 TLS handshake support

4.14.1 TLSGenerateRandom

Generates a random that is stored in the A5000 and used by TLSPerformPRF.

Table 209. 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 210. TLSGenerateRandom R-APDU Body

Value	Description
TLV[TAG_1]	32-byte random value

Table 211. TLSGenerateRandom R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.14.2 TLSCalculatePreMasterSecret

The command TLSCalculatePreMasterSecret will compute the pre-master secret for TLS according to [RFC5246]. The pre-master secret will always be stored in an HMACKey object (TLV[TAG_3]). The HMACKey object must be created before with the expected length of the pre master secret; otherwise the calculation of the pre-master secret will fail.

Supported algorithms and related input data are listed in following table:

Table 212. Supported TLS 1.2 configurations

Config	RFC reference	PSK (TLV[TAG_ 1])	ECKey key pair (TLV{TAG_2])	Input data (TLV[TAG_4])
PSK Key Exchange	[RFC4279]	v		none
ECDHE_PSK Key Exchange	[RFC5489]	v	v	external EC public key
EC Key Exchange	[RFC4492]		v	external EC public key

When POLICY_OBJ_ALLOW_DERIVED_INPUT is applied to prevent write access to the target object, this policy must have either the ECKey key pair as extension. If no key pair is present, the extension must contain the identifier of the PSK.

Table 213. 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 214. TLSCalculatePreMasterSecret R-APDU Body

Value	Description
-	

Table 215. TLSCalculatePreMasterSecret R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.14.3 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. Note that the use of TLSPerformPRF for key expansion requires to have P2 equal to P2_PRF_BOTH as the user must be able to insert both random values.

Each time before calling this function, TLSGenerateRandom must be called. Executing this function will clear the random that is stored in the A5000.

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.

This results in P2 having these possibilities:

- P2_TLS_PRF_CLI_HELLO: pass the clientHelloRandom to calculate a master secret, the serverHelloRandom is in A5000, generated by TLSGenerateRandom.
- P2_TLS_PRF_SRV_HELLO: pass the serverHelloRandom to calculate a master secret, the clientHelloRandom is in A5000, generated by TLSGenerateRandom.
- P2_TLS_PRF_CLI_RANDOM: pass the clientRandom to generate key expansion data, the serverRandom is in A5000, generated by TLSGenerateRandom.
- P2_TLS_PRF_SRV_RANDOM: pass the serverRandom to generate key expansion data, the clientRandom is in A5000
- P2_PRF_BOTH: pass the clientRandom and serverRandom (in the order that the user defines) to calculate a master secret or key expansion data. In this case, the input HMAC key must have the policy POLICY_OBJ_ALLOW_TLS_KDF_EXT_RANDOM set. Also note that the policy should in general be allowed: if extended feature bit EXTCFG_CRYPTO_TLS_KDF_ALLOW_EXT_RANDOM_POLICY is not set, this policy cannot be applied to any object, hence P2_PRF_BOTH can not be used succesfully.

Table 216. 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. Minimum policy:POLICY_OBJ_ALLOW_TLS_KDF Optional policy: POLICY_OBJ_ALLOW_TLS_KDF_EXT_RANDOM (see description above). Optional policy: POLICY_OBJ_FORBID_DERIVED_OUPUT would prevent to execute this command succesfully.

Table 216. TLSPerformPRF C-APDU*...continued*

Field	Value	Description
	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 or 64-byte random value (any P2 except P2_PRF_BOTH requires 32 bytes; P2_PRF_BOTH requires 64 bytes).
	TLV[TAG_5]	2-byte requested length (1 up to 512 bytes)
Le	0x00	

Table 217. TLSPerformPRF R-APDU Body

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

Table 218. TLSPerformPRF R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.15 Digest operations

There are 2 options to use Digest operations on A5000:

- 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.

4.15.1 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 219. 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 220. DigestInit R-APDU Body

Value	Description
-	

Table 221. DigestInit R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.15.2 DigestUpdate

Update a digest operation.

Table 222. DigestUpdate C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	See Instruction
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 223. DigestUpdate R-APDU Body

Value	Description
-	-

Table 224. DigestUpdate R-APDU Trailer

SW	Description
SW_NO_ERROR	The command is handled successfully.

4.15.3 DigestFinal

Finalize a digest operation.

Table 225. DigestFinal C-APDU

Field	Value	Description
CLA	0x80	
INS	INS_CRYPTO	See Instruction
P1	P1_DEFAULT	See P1
P2	P2_FINAL	See P2

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

Field	Value	Description
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 226. DigestFinal R-APDU Body

Value	Description
TLV[TAG_1]	hash value

Table 227. DigestFinal R-APDU Trailer

SW	Description
SW_NO_ERROR	The hash is created successfully.

4.15.4 DigestOneShot

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

Table 228. 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 229. DigestOneShot R-APDU Body

Value	Description
TLV[TAG_1]	Hash value.

Table 230. DigestOneShot R-APDU Trailer

SW	Description
SW_NO_ERROR	The hash is created successfully.

4.16 Generic management commands

4.16.1 GetVersion

Gets the Application version information.

This will return 7-byte or 37-byte VersionInfo (including major, minor and patch version of the Application, supported Application features and secure box version).

Table 231. 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
Lc	#(Payload)	
Le	0x00	Expecting TLV with 7-byte data (when P2 = P2_VERSION) or a TLV with 37 byte data (when P2= P2_VERSION_EXT).

Table 232. GetVersion R-APDU Body

Value	Description
TLV[TAG_1]	7-byte VersionInfo (if P2 = P2_VERSION) or 7-byte VersionInfo followed by 30 bytes extendedFeatureBits (if P2 = P2_VERSION_EXT)

Table 233. GetVersion R-APDU Trailer

SW	Description
SW_NO_ERROR	Data is returned successfully.

4.16.2 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 234. 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 235. GetTimestamp R-APDU Body

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

Table 236. GetTimestamp R-APDU Trailer

SW	Description
SW_NO_ERROR	Data is returned successfully.

4.16.3 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 Application 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 237. 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 238. 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 239. GetFreeMemory R-APDU Trailer

SW	Description
SW_NO_ERROR	Data is returned successfully.

4.16.4 GetRandom

Gets random data from the A5000.

Table 240. 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 241. GetRandom R-APDU Body

Value	Description
TLV[TAG_1]	Random data.

Table 242. GetRandom R-APDU Trailer

SW	Description
SW_NO_ERROR	Data is returned successfully.

4.16.5 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 243. 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 244. DeleteAll R-APDU Body

Value	Description
-	

Table 245. 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 246. APDU list

Name	CLA	INS	P1	P2	Remarks
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	
SCPInitializeUpdate	0x80	0x50			
SCPExternalAuthenticate	0x80	0x82			
ECKeySessionInternalAuthenticate	0x80	0x88	0x00	0x00	
SetLockState	0x80	0x04	0x00	0x3E	Protected by RESERVED_ID_TRANSPORT (if present).
DisableSecureObjectCreation	0x80	0x04	0x00	0x57	Protected by RESERVED_ID_RESTRICT (if present).
SetPlatformSCPRequest	0x80	0x04	0x00	0x52	Protected by RESERVED_ID_PLATFORM_SCP (if present).
SetApplicationFeatures	0x80	0x04	0x00	0x3F	Protected by RESERVED_ID_FEATURE (if present).
SendCardManagerCommand	0x80	0x04	0x00	0x55	
TriggerSelfTest	0x80	0x04	0x00	0x58	
ReadState	0x80	0x02	0x00	0x5B	
WriteECKey	0x80	0x01*	key type | 0x01	0x00	* can be in addition: INS_TRANSIENT (0x80), INS_AUTH_OBJECT (0x40) and INS_ATTEST (0x20).
WriteSymmKey	0x80	0x01*	Type	0x00	* can be in addition: INS_TRANSIENT (0x80), INS_AUTH_OBJECT (0x40) and INS_ATTEST (0x20).
WriteBinary	0x80	0x01*	0x06	0x00	* can be in addition: INS_TRANSIENT (0x80) and INS_ATTEST (0x20).
WriteUserID	0x80	0x01*	0x07	0x00	* can be in addition: INS_TRANSIENT (0x80), INS_AUTH_OBJECT (0x40) and INS_ATTEST (0x20).
WriteCounter	0x80	0x01*	0x08	0x00	* can be in addition: INS_TRANSIENT (0x80) and INS_ATTEST (0x20).
WritePCR	0x80	0x01	0x09	0x00	

Table 246. APDU list*...continued*

Name	CLA	INS	P1	P2	Remarks
ImportObject	0x80	0x01	0x00	0x18	
ImportExternalObject	0x80	0x06	0x00	0x00	
ReadObject	0x80	0x02	0x00	0x00	
ReadAttributes	0x80	0x02	0x00	0x3B	
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	
CreateCryptoObject	0x80	0x01	0x10	0x00	
ReadCryptoObjectList	0x80	0x02	0x10	0x25	
DeleteCryptoObject	0x80	0x04	0x10	0x28	
ECDSASign	0x80	0x03	0x0C	0x09	
ECDSAVerify	0x80	0x03	0x0C	0x0A	
ECDHGenerateSharedSecret	0x80	0x03	0x01	0x0F or 0x59	
EC Point Multiplication	0x80	0x03	0x01	0x62	
CipherInit	0x80	0x03	0x0E	0x42 or 0x43	
CipherUpdate	0x80	0x03	0x0E	0x0C	
CipherFinal	0x80	0x03	0x0E	0x0D	
CipherOneShot	0x80	0x03	0x0E	0x37 or 0x38	
AEADInit	0x80	0x03	0x11	0x42 or 0x43	
AEADUpdate	0x80	0x03	0x11	0x0C	
AEADFinal	0x80	0x03	0x11	0x0D	
AEADOneShot	0x80	0x03	0x11	0x37 or 0x38	
MACInit	0x80	0x03	0x0D	0x03	
MACUpdate	0x80	0x03	0x0D	0x0C	
MACFinal	0x80	0x03	0x0D	0x0D	

Table 246. APDU list*...continued*

Name	CLA	INS	P1	P2	Remarks
MACOneShot	0x80	0x03	0x0D	0x45/0x46	
HKDFExtractAndExpand	0x80	0x03	0x00	0x2D	
HKDFExpandOnly	0x80	0x03	0x00	0x2F	
TLSGenerateRandom	0x80	0x03	0x0F	0x49	
TLSCalculatePreMasterSecret	0x80	0x03	0x0F	0x4A	
TLSPerformPRF	0x80	0x03	0x0F	0x4B-0x4E or 0x5A	
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 or 0x21	
GetTimestamp	0x80	0x04	0x00	0x3D	
GetFreeMemory	0x80	0x04	0x00	0x22	
GetRandom	0x80	0x04	0x00	0x49	
DeleteAll	0x80	0x04	0x00	0x2A	

6 Policy mapping

6.1 Policy mapping tables

6.1.1 Policy mapping to symmetric key Secure Objects

The table below uses the following syntax: "v" means supported, empty cells mean not supported; A = Authentication Object; NA= Non-Authentication Object

Table 247. Policy mapping SymmKey Secure Objects

policy (starting with "POLICY_OBJ_")	Function	AESKey	DESKey	HMACKey	TLV	Description	A	NA
ALLOW_TLS_KDF	TLSPerformPRF			v	TAG_1	input key		v
ALLOW_TLS_PMS	TLSCalculatePreMasterSecret			v	TAG_1	PSK (unless ALLOW_WRITE is set).		v
ALLOW_SIGN	MACInit	v		v	TAG_1	input key		v
	MACOneShot	v	v	v	TAG_1	input key		v
ALLOW_VERIFY	MACInit	v		v	TAG_1	input key		v
	MACOneShot	v	v	v	TAG_1	input key		v
ALLOW_ENC	CipherInit	v	v		TAG_1	input key		v
	CipherOneShot	v	v		TAG_1	input key		v
	AEADInit	v			TAG_1	input key		v
	AEADOneShot	v			TAG_1	input key		v
ALLOW_DEC	CipherInit	v	v		TAG_1	input key		v
	CipherOneShot	v	v		TAG_1	input key		v
	AEADInit	v			TAG_1	input key		v
	AEADOneShot	v			TAG_1	input key		v
ALLOW_HKDF	HKDFExtractAndExpand			v	TAG_1	IKM		v
				v	TAG_6	salt (if present)		v
	HKDFExpandOnly			v	TAG_1	PRK		v
				v	TAG_6	IV (if present)		v

All information provided in this document is subject to legal disclaimers.

Table 247. Policy mapping SymmKey Secure Objects*...continued*

policy (starting with "POLICY_OBJ_")	Function	AESKey	DESKey	HMACKey	TLV	Description	A	NA
ALLOW_RFC3394_ UNWRAP	WriteSymmKey	v			TAG_3	Key Encryption Key		v
ALLOW_READ	ReadObject	v	v	v	TAG_1	Object to read (for SymmKeys, this only works when attestation is requested and this will not return the key value)	v	v
ALLOW_WRITE	WriteSymmKey	v	v	v	TAG_1	Object to write (policy only applies when the object already exists)	v	v
ALLOW_DELETE	DeleteSecureObject	v	v	v	TAG_1	Object to delete (only when the Secure Object does not have ORIGIN_ PROVISIONED).	v	v
REQUIRE_SM	(any)	v	v	v	N.A.	Any access to the object requires secure messaging, at least C-MAC.	v	v
REQUIRE_PCR_VALUE	(any)	v	v	v	N.A.	Any access to the object requires a matching PCR value.	v	v
ALLOW_IMPORT_EXPORT	ExportObject	v	v	v	TAG_1	transient object to export from		v
	ImportObject	v	v	v	TAG_1	transient object to import to		v
FORBID_DERIVED_ OUTPUT	CipherInit	v	v		TAG_1	input key		v
	CipherOneShot	v	v		TAG_1	input key		v
	AEADInit	v			TAG_1	input key		v
	AEADOneShot	v			TAG_1	input key		v
	MACInit	v		v	TAG_1	input key		v
	MACOneShot	v	v	v	TAG_1	input key		v
	HKDFExtractAndExpand			v	TAG_1	IKM		v
				v	TAG_6	salt		v
	HKDFExpandOnly			v	TAG_1	PRK		v
				v	TAG_6	salt		v
	TLSPerformPRF			v	TAG_1	input key		v

Table 247. Policy mapping SymmKey Secure Objects*...continued*

policy (starting with "POLICY_OBJ_")	Function	AESKey	DESKey	HMACKey	TLV	Description	A	NA
ALLOW_TLS_KDF_EXT_ RANDOM	TLSPerformPRF			v	TAG_1	input key		v
ALLOW_DERIVED_INPUT	ECDHGenerateSharedSecret	v		v	TAG_7	target output object		v
	HKDFExtractAndExpand	v		v	TAG_7	input key		v
	HKDFExpandOnly	v		v	TAG_7	input key		v
	TLSCalculatePreMasterSecret			v	TAG_3	target output object		v
FORBID_EXTERNAL_IV	CipherInit	v	v		TAG_1	input key		v
	CipherOneShot	v	v		TAG_1	input key		v
	AEADInit	v	v		TAG_1	input key		v
	AEADOneShot	v	v		TAG_1	input key		v
ALLOW_USAGE_AS_ HMAC_PEPPER	HKDFExtractAndExpand			v	TAG_6	salt (if present) and no target output object given.		v
	HKDFExpandOnly			v	TAG_6	salt (if present) and no target output object given.		v

6.1.2 Policy mapping to ECKey Secure Objects

The table below uses the following syntax: "v" means supported, empty cells mean not supported; A = Authentication Object; NA= Non-Authentication Object

Table 248. Policy mapping ECKey Secure Objects

policy (starting with "POLICY_OBJ_")	Function	EC Keypair	EC public key	EC private key	TLV	Description	A	NA
ALLOW_SIGN	ECDSASign	v		v	TAG_1	private key		v
ALLOW_VERIFY	ECDSAVerify	v	v		TAG_1	public key		v
ALLOW_KA	ECDHGenerateSharedSecret	v		v	TAG_1	input key object		v
	ECPointMultiply	v		v	TAG_1	input key object		v
	TLSCalculatePreMasterSecret	v			TAG_2	key pair in PSK_ECDHE or ECDHE.		v
ALLOW_ENC	-							

All information provided in this document is subject to legal disclaimers.

Table 248. Policy mapping ECKey Secure Objects*...continued*

policy (starting with "POLICY_OBJ_")	Function	EC Keypair	EC public key	EC private key	TLV	Description	A	NA
ALLOW_DEC	-							
ALLOW_READ	ReadObject	v	v	v	TAG_1	key to read (this will only return the public key value).	v	v
ALLOW_WRITE	WriteECKey	v	v	v	TAG_1	key to write	v	v
ALLOW_GEN	WriteECKey	v		v	TAG_1	key pair or private key to generate		v
ALLOW_DELETE	DeleteSecureObject	v	v	v	TAG_1	Object to delete (only when the Secure Object does not have ORIGIN_ PROVISIONED).	v	v
REQUIRE_SM	(any)	v	v	v	N.A.	Any access to the object requires secure messaging, at least C-MAC.	v	v
REQUIRE_PCR_VALUE	(any)	v	v	v	N.A.	Any access to the object requires a matching PCR value.	v	v
ALLOW_ATTESTATION	ReadObject	v		v	TAG_5	attestating key		v
	TriggerSelfTest	v		v	TAG_6	attestating key		v
ALLOW_IMPORT_EXPORT	ExportObject	v	v	v	TAG_1	transient object to export from		v
	ImportObject	v	v	v	TAG_1	transient object to import to		v
FORBID_DERIVED_ OUTPUT	ECDSASign	v		v	TAG_1	input key object		v
	ECDHGenerateSharedSecret	v		v	TAG_1	input key object		v
	ECPointMultiply	v		v	TAG_1	input key object		v
ALLOW_DERIVED_INPUT	ECPointMultiply		v		TAG_7	target output object		v
INTERNAL_SIGN	ECDSASign	v		v	TAG_1	key to sign		v

6.1.3 Policy mapping to File Secure Objects

The table below uses the following syntax: "v" means supported, empty cells mean not supported; A = Authentication Object; NA= Non-Authentication Object

Table 249. Policy mapping

policy (starting with "POLICY_OBJ_")	Function	Binary file	UserID	Counter	PCR	TLV	Description	A	NA
ALLOW_SIGN	-								
ALLOW_VERIFY	-								
ALLOW_KA	-								
ALLOW_ENC	-								
ALLOW_DEC	-								
ALLOW_RFC3394_UNWRAP	-								
ALLOW_READ	ReadObject	v	v	v	v	TAG_1	object to read.		
ALLOW_WRITE	WriteBinary	v				TAG_1	BinaryFile to be updated		
	WriteCounter			v		TAG_1	Counter to be updated.		
	WritePCR				v	TAG_1	PCR to be updated.		
ALLOW_DELETE	DeleteSecureObject	v	v	v	v	TAG_1	Object to be deleted.		
	WritePCR				v	TAG_1	PCR to be reset		

Table 249. Policy mapping*...continued*

policy (starting with "POLICY_OBJ_")	Function	Binary file	UserID	Counter	PCR	TLV	Description	A	NA
REQUIRE_SM	(any)	v	v	v	v	N.A.	Any access to the object requires secure messaging, at least C-MAC.	v	v
REQUIRE_PCR_VALUE	(any)	v		v	v	N.A.	Any access to the object requires a matching PCR value.	v	v
ALLOW_ATTESTATION	-								
ALLOW_DESFIRE_AUTHENTICATION	-								
ALLOW_DESFIRE_DUMP_SESSION_KEY	-								
ALLOW_IMPORT_EXPORT	-						transient object to export from		v

Table 249. Policy mapping*...continued*

policy (starting with "POLICY_OBJ_")	Function	Binary file	UserID	Counter	PCR	TLV	Description	A	NA
	-						transient object to import to		v
FORBID_DERIVED_OUTPUT	-								
ALLOW_DESFIRE_CHANGEKEY	-								
ALLOW_DERIVED_INPUT	-								

7 Example sequences

7.1 AES GCM/GMAC

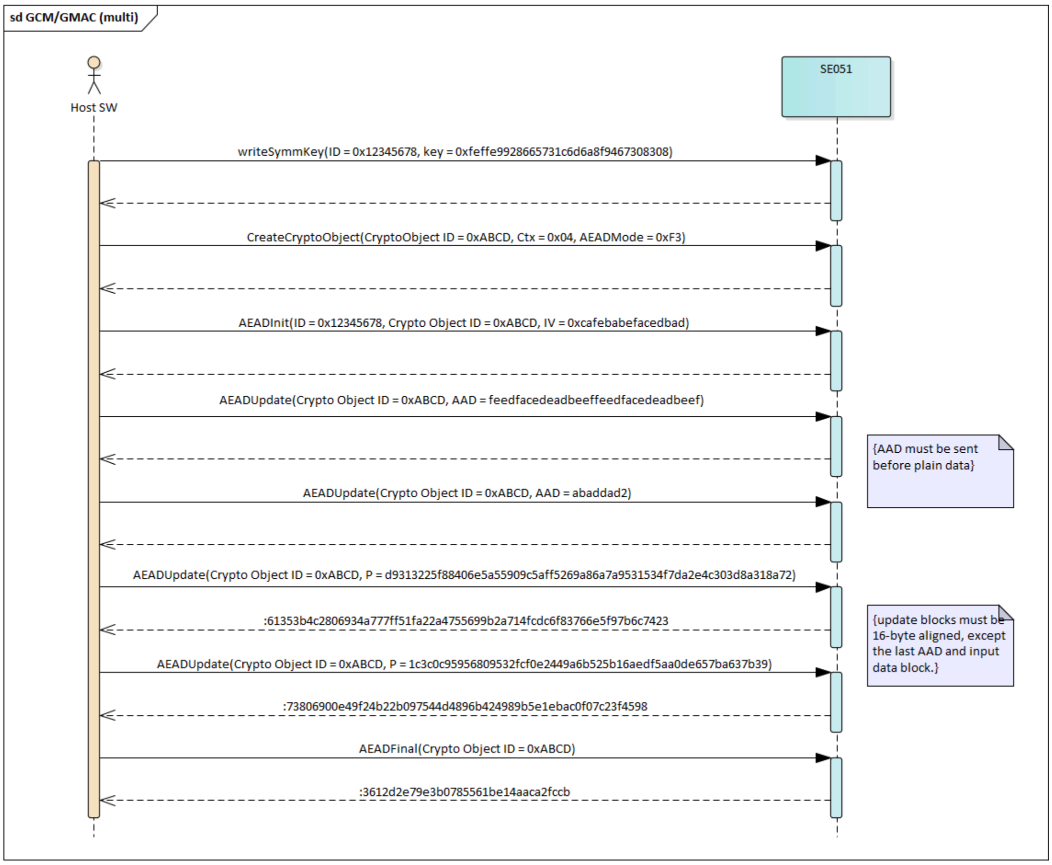
sd GCM/GMAC (multi)
Host SW
SE051
writeSymmKey(ID = 0x12345678, key = 0xfeffe9928665731c6d6a8f9467308308)
CreateCryptoObject(CryptoObject ID = 0xABCD, Ctx = 0x04, AEADMode = 0xF3)
AEADInit(ID = 0x12345678, Crypto Object ID = 0xABCD, IV = 0xcafebabefacedbad)
AEADUpdate(Crypto Object ID = 0xABCD, AAD = feedfacedeadbeeffeedfacedeadbeef)
{AAD must be sent before plain data}
AEADUpdate(Crypto Object ID = 0xABCD, AAD = abaddad2)
AEADUpdate(Crypto Object ID = 0xABCD, P = d9313225f88406e5a55909c5aff5269a86a7a9531534f7da2e4c303d8a318a72)
:61353b4c2806934a777ff51fa22a4755699b2a714fcdc6f83766e5f97b6c7423
{update blocks must be 16-byte aligned, except the last AAD and input data block.}
AEADUpdate(Crypto Object ID = 0xABCD, P = 1c3c0c95956809532fcf0e2449a6b525b16aedf5aa0de657ba637b39)
:73806900e49f24b22b097544d4896b424989b5e1ebac0f07c23f4598
AEADFinal(Crypto Object ID = 0xABCD)
:3612d2e79e3b0785561be14aaca2fccb

Figure 16. Example GCM operation in multiple steps (P2_ENCRYPT)

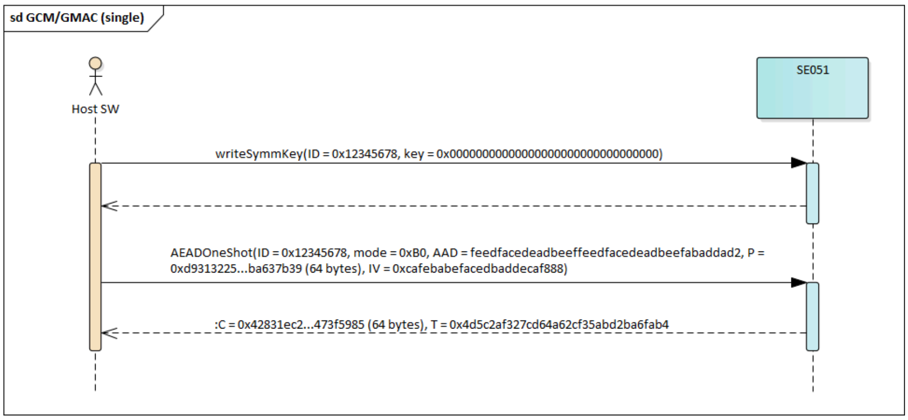
sd GCM/GMAC (single)
Host SW
SE051
writeSymmKey(ID = 0x12345678, key = 0x00000000000000000000000000000000)
AEADOneShot(ID = 0x12345678, mode = 0xB0, AAD = feedfacedeadbeeffeedfacedeadbeefabaddad2, P = 0xd9313225...ba637b39 (64 bytes), IV = 0xcafebabefacedbaddecaf888)
:C = 0x42831ec2...473f5985 (64 bytes), T = 0x4d5c2af327cd64a62cf35abd2ba6fab4

Figure 17. Example GCM operation in one shot mode (P2_ENCRYPT)

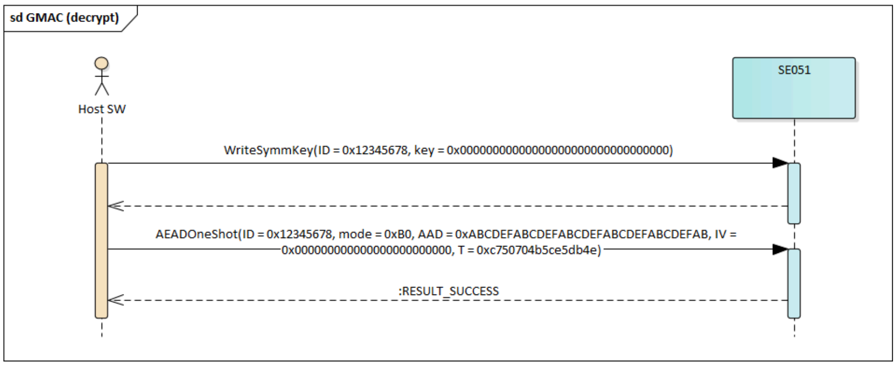
sd GMAC (decrypt)
Host SW
SE051
WriteSymmKey(ID = 0x12345678, key = 0x00000000000000000000000000000000)
AEADOneShot(ID = 0x12345678, mode = 0xB0, AAD = 0xABCDEFABCDEFABCDEFABCDEFABCDEFAB, IV = 0x000000000000000000000000, T = 0xc750704b5ce5db4e)
:RESULT_SUCCESS

Figure 18. Example GMAC operation in multiple steps (P2_DECRYPT)

8 Memory consumption

8.1 Secure Objects

This section shows the amount of memory taken by 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 250. Secure Object memory ECKey

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 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

Table 251. Secure Object memory SymmKey

Object Type	Persistent key [bytes]	Transient key [bytes]
AESKey	NVM: 136 + key size in bytes RAM: 0	NVM: 116 RAM: 16 + key size in bytes
DESKey	NVM: 160 + key size in bytes RAM: 0	NVM: 136 RAM: 32 + key size in bytes
HMACKey	NVM: 140 + key size in bytes RAM: 0	NVM: 120 RAM: 16 + key size in bytes

Table 252. Secure Object memory File objects

Object Type	Persistent object [bytes]	Transient object [bytes]
BinaryFile	NVM: 96 + file size in bytes RAM: 0	NVM: 92 RAM: file size in bytes
Counter	NVM: 96 + counter size in bytes RAM: 0	NVM: 92 RAM: 16
PCR	NVM: 180 RAM: 0	NVM: 144 RAM: 32
UserID	NVM: 112 RAM: 0	Not Applicable

8.2 Crypto Objects

Table 253. 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	224
Digest	DIGEST_SHA512	108	224
Cipher	DES_CBC_NOPAD	116	32
Cipher	DES_CBC_ISO9797_M1	116	32
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	48
Cipher	AES_CBC_ISO9797_M1	116	48
Cipher	AES_CBC_ISO9797_M2	116	48
Cipher	AES_CBC_PKCS5	116	32
Cipher	AES_CTR	116	48
Signature	HMAC_SHA1	112	240
Signature	HMAC_SHA256	112	288
Signature	HMAC_SHA384	112	560
Signature	HMAC_SHA512	112	560
Signature	CMAC_128	116	48
AEAD	AES_GCM	124	96
AEAD	AES_CCM	280	160

9 Abbreviations

AEAD	Authenticated Encryption with Associated Data
AES	Advanced Encryption Standard
API	Application Programming Interface
APDU	Application Protocol Data Unit
CCM	Counter with CBC-MAC
CLA	Class
DES	Data Encryption Standard
EC	Elliptic Curve
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie Hellman
ECKA	Elliptic Curve Key Agreement
GCM	Galois Counter Mode
GMAC	Galois 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
PCR	Platform Configuration Register
PICC	Proximity IntegratedCircuit Card
PRF	Pseudo Random Function
PSK	Pre Shared Key
Rev	Revision
SCP	Secure Channel Protocol
SSD	Supplementary Security Domain
TLS	Transport Layer Security
TLV	Tag Length Value
UGM	User Guidance Manual

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] **[AN10922]** — Symmetric Key Diversifications
NXP Semiconductors
https://www.nxp.com/docs/en/application-note/AN10922.pdf

[6] **[UserGuidelines]** — A5000 - User Guidelines
NXP Semiconductors
Document number: AN13266
https://www.docstore.nxp.com

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.

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.

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.

Non-automotive qualified products — Unless this data sheet expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications. In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Evaluation products — This product is provided on an "as is" and "with all faults" basis for evaluation purposes only. NXP Semiconductors, its affiliates and their suppliers expressly disclaim all warranties, whether express, implied or statutory, including but not limited to the implied warranties of non-infringement, merchantability and fitness for a particular purpose. The entire risk as to the quality, or arising out of the use or performance, of this product remains with customer. In no event shall NXP Semiconductors, its affiliates or their suppliers be liable to customer for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out the use of or inability to use the product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages. Notwithstanding any damages that customer might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP Semiconductors, its affiliates and their suppliers and customer's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by customer based on reasonable reliance up to the greater of the amount actually paid by customer for the product or five dollars (US$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose.

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.

MIFARE — 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

Please be aware that important notices concerning this document and the product(s) described herein, have been included in section 'Legal information'.

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

Date of release: 28 March 2022