**Demo Applet**

Service Specification

Date

Change History

|  |  |  |  |
| --- | --- | --- | --- |
| **Description** | **Date** | **Author** | **Reviewer** |
| Demo Applet Service Description | May 2014 | Md Mazharul Islam  Abdullah Al-Shamim | Abdullah Al-Shamim |
|  |  |  |  |
|  |  |  |  |

**First Release (v1.0.0) Features:**

* Authentication Service: using GP SCP02 protocol with 2key TDES. Authentication must be done to perform any kind of operation on card
* Update Demo Key Service:
  + Updates DEM-AUTH key container with 2key TDES
  + Updates DEM-KEY-WRAP container with AES-128/192/256 or TDES 2/3key
  + Updates DEM-MAC key container with TDES 2/3key or AES-128/256 or HMAC key with block length 128
  + Updates DEM-3P-PUB key container with RSA public key(length 1024 or 2048) or EC public key (using curve P-192, P-224, P-256, P-384, or P-521)
* Key-Pair Generation Service:
  + Generates RSA key-pair of length 2048and EC key-pair using curve P-224, P-256, P-384, or P-521
  + No public key is returned during key generation
  + Public key is returned by another apdu (GET DATA)
* Random# Generation Service: Generates random number with specified length and seed value
* Digital Signature Service:
  + Performs sign with RSA (2048) and EC (curve: P-224, P-256, P-384, or P-521) private key
  + Performs verify with RSA (1024 or 2048) and EC (curve: P-192, P-224, P-256, P-384, or P-521) public key
* Message Authentication Service:
  + Generates and verifies MAC with TDES(2/3 keys)
  + Generates and verifies MAC with AES-128 (FIPS Approved AES-256 but not supported by KONA2 N41M0 card)
  + Generates and verifies MAC with HMAC key with SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512
* Key Agreement Service:
  + Generates shared secret using ECHD with curve P-224, P-256, P-384, or P-521
  + Right now the generated shared secret is not used for any other purposes
* Message Digest service:Generates message digest with SHA-1 and SHA-2 (SHA-224, SHA-256, SHA-384, and SHA-512)
* Key Wrap Service:
  + Wrap and unwrap using TDES 2/3key
  + Wrap and unwrap using AES-128/192/256 key
* Destroy CSPs Service: Destroys/zeroize all **DemoApplet** keys and CSPs except Authentication key which is rewrite by a default base key value

# System Requirements

* Java Card with API 3.0.4 implemented
* Memory Requirement: Total memory required in E2P ROM is 24KB (Load File 9KB+ Instance15KB) (apx.)

# Outstanding issues

* Right now no state is maintain for the **DemoApplet**
* Right now no PIN authentication is provided
* SCP03 is not supported
* Key wrap/unwrap service now is a standalone service
* Signature generation and verification with P-521 is not perfectly ok with off card java application as in java application signature length is specified by two bytes but in java card it is specified by one byte
* Right now shared secret generated from ECDH key agreement is not used elsewhere

Table of Contents

[System Requirements 3](#_Toc387929479)

[Outstanding issues 3](#_Toc387929480)

[1 Introduction 6](#_Toc387929481)

[2 Authenticate Service 7](#_Toc387929482)

[3 Update Demo Keys Service 7](#_Toc387929483)

[3.1 Update DEM-AUTH Key Container 7](#_Toc387929484)

[3.2 Update DEM-MAC Key Container 8](#_Toc387929485)

[3.3 Update DEM-KEY-WRAP Container 8](#_Toc387929486)

[3.4 Update DEM-3P-PUB key containers 9](#_Toc387929487)

[3.1.1 Load RSA public key: 9](#_Toc387929488)

[3.1.2 Load EC public key: 10](#_Toc387929489)

[4 Random Number Generation Service 11](#_Toc387929490)

[5 Key Pair Generation Service 11](#_Toc387929491)

[5.1 RSA key pair: 11](#_Toc387929492)

[5.2 EC key pair: 12](#_Toc387929493)

[6 GET Data Service 12](#_Toc387929494)

[7 Manage Security Environment (MSE) Service 13](#_Toc387929495)

[7.1 Restore 13](#_Toc387929496)

[7.2 SET 14](#_Toc387929497)

[8 Digital Signature Service 15](#_Toc387929498)

[8.1 Signature generation 15](#_Toc387929499)

[8.2 Signature verification 16](#_Toc387929500)

[9 Message Authentication Service 18](#_Toc387929501)

[9.1 MAC generation 18](#_Toc387929502)

[9.2 MAC verification 19](#_Toc387929503)

[10 Message Digest Service 21](#_Toc387929504)

[11 Key Wrap Service 22](#_Toc387929505)

[11.1 Wrap: 22](#_Toc387929506)

[11.2 Unwrap: 22](#_Toc387929507)

[12 Key Agreement Service 23](#_Toc387929508)

[13 Destroy Service 23](#_Toc387929509)

# Introduction

This document describes all of the services in **DemoApplet**. The services have some dependency on each other and these dependencies also describe here.

The command set of the applet is based on ISO/IEC 7816-4, ISO/IEC 7816-8, and some proprietary command based on these standards.

The **DemoApplet** uses Java Card v3.0.4 and Global Platform v2.2.1 APIs as the development platform.

The main purpose of this Applet is to perform many of the basic crypto operations such as Wrap/Un-wrap, Signature/Verify, Hash and Mac. It supports five different types of keys: RSA, DES, AES, EC and HMAC keys of different length and corresponding many algorithms discussed later in the documentation. It contains four key containers to load authentication key (DEM-AUTH), wrap key (DEM-WRAP), MAC key (DEM-MAC) and public key from off-card entity. Key-pair generated by **DemoApplet** for RAS and EC key stored in separate container and can be read by off-card entity later.

All the algorithm/mechanism used in different **DemoApplet** services is FIPS approved.

Prior to invoke any **DemoApplet** service authentication must be done first using GP SCP02.

List of services that are provided by **DemoApplet**:

* Authentication service
* Update Demo Key service
* Random Number Generation service
* Key Pair Generation service
* Digital Signature service
* Message Authentication service
* Message Digest service
* Key Wrap service
* Key Agreement service
* Destroy Demo Applet CSPs service

# Authenticate Service

Authenticate Service makes a secure channel between on-card and off-card. **DemoApplet** use GP’s SCP-02 for Authenticate Service. Without authentication the applet user cannot perform any operation in **DemoApplet.** The Authentication service is performed through two separate commands as specified in GP 2.2.1:

1. Initialize update.
2. External authenticate.

**Supported keys**: Triple DES 2Key, initially the base key for external authentication is 16 byte constant value contains in DEM-AUTH key container. But it can be updated by Update DEM-AUTH Key Container service.

See Demo Applet APDU specification for commands detail.

# Update Demo Keys Service

This service is for put data into key containers. Demo Applet has four Key containers to contain different types of keys:

1. DEM-AUTH key container
2. DEM-MAC key container
3. DEM-KEY-WRAP key container
4. DEM-3P-PUB key container

Update demo key service is for put keys into these Key Containers. Update Demo Keys Service supports different types of keys: RSA public key, EC public key, DES Key, AES Key, and HMAC key to put into key containers.

To perform different types of crypto operation, the appropriate key will be put into these key containers according to this service. See Demo Applet APDU specification for the APDU structure of Update Demo Keys Service.

## Update DEM-AUTH Key Container

To perform Authentication the base key can be put into DEM-AUTH Key container through this service.

**Supported key**: TDES 2Key

**Steps to follow for updating DEM-AUTH Key Container:**

1. Perform Authentication (see section: **Error! Reference source not found.**).
2. Use Update Demo Keys command as specified in “Demo Applet APDU Specification” document for loading the new base key for authentication.

Sample APDU:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| CLA | INS | P1 | P2 | LC | Data | LE |
| 00 | DA | 01 | 01 | 10 | 16 byte DES key | -- |

## 3.2 Update DEM-MAC Key Container

To generate and verify MAC (See section: Message Authentication Service), the appropriate MAC key is needed which can be load into DEM-MAC key container. This service loads the MAC keys into DEM-MAC key container.

**Supported keys**:

* DES key (16 or 24 bytes).
* AES-128 key (16 bytes).
* HMAC key (Not more than 64 bytes).

**Steps to follow for updating DEM-MAC Key Container:**

1. Perform Authentication (see section: **Error! Reference source not found.**).
2. Use Update Demo Keys command as specified in “Demo Applet APDU Specification” document for loading the MAC key.

Sample APDU for loading AES-128 key:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| CLA | INS | P1 | P2 | LC | Data | LE |
| 00 | DA | 01 | 22 | 10 | 16 byte AES key | -- |

## 3.3 Update DEM-KEY-WRAP Container

To perform Key Wrap Service the appropriate Wrap or Unwrap key is needed which can be load into DEM-KEY-WRAP key container. ‘Update DEM-KEY-WRAP Container’ service loads those keys into DEM-KEY-WRAP key container.

**Supported keys**:

* DES key (16 or 24 bytes).
* AES key (16, 24 or 32 bytes).

**Steps to follow for updating DEM-KEY-WRAP Key Container:**

1. Perform Authentication (see section: **Error! Reference source not found.**).
2. Use Update Demo Keys command as specified in “Demo Applet APDU Specification” document for loading the Wrap/Unwrap key.

Sample APDU for loading Wrap-AES-256 key:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| CLA | INS | P1 | P2 | LC | Data | LE |
| 00 | DA | 01 | 13 | 20 | 32 byte AES key | -- |

## 3.4 Update DEM-3P-PUB key containers

For Digital signature verification (See section: ) and Key Agreement (See section: Key Agreement Service) service, appropriate public key is needed which can be put into DEM-3P-PUB key containers. This service loads the public keys into DEM-3P-PUB key container.

**Supported keys**:

* RSA public key (both 1024 and 2048 bits). Only for sign verification.
* EC public key (all EC keys approved by NIST and FIPS). Both sign verification and key agreement.

Steps to follow for updating DEM-3P-PUB Key Container:

1. Perform Authentication (see section: **Error! Reference source not found.**).
2. Use Update Demo Keys command as specified in “Demo Applet APDU Specification” document for loading the public key.

### Load RSA public key:

To load RSA-1024 public key two APDU command is needed.

* One’s for putting modulus of the keys.
* And the other for exponent!

But in RSA-2048, three APDU command is needed.

* First one is for putting the first part of modulus of the keys (1st 128 bytes) staring with a tag “81”.
* The second one is for putting the last part of modulus of the keys (2nd 128 bytes) staring with a tag “82”.
* And the last APDU is for exponent!

In every case the modulus of the public key will be put first into the key container than the exponent. User cannot put exponent first if the modulus is not already put into the key container.

For RSA-2048, 2nd part of the modulus cannot be put into the key container if the 1st part of the modulus is not put yet into the key container.

The public key cannot be initialized properly if the exponent is not put into the key container. Hence no crypto operation can be performed by this public key!

Sample APDU for loading RSA-1024 public key modulus:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| CLA | INS | P1 | P2 | LC | Data | LE |
| 00 | DA | 01 | 31 | 80 | 128 bytes RSA modulus | -- |

Sample APDU for loading RSA-1024 public key exponent:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| CLA | INS | P1 | P2 | LC | Data | LE |
| 00 | DA | 01 | 31 | Exponent length | Public key exponent | -- |

### Load EC public key:

The supported EC public keys to load into DEM-3P-PUB key container are:

1. EC P-192.
2. EC P-224.
3. EC P-256.
4. EC P-384.
5. EC P-521.

A single APDU is needed to load EC public key into key container.

Sample APDU for loading EC public key for Key Agreement service:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| CLA | INS | P1 | P2 | LC | Data | LE |
| 00 | DA | 01 | 36 | Public key length | Publick key data | -- |

For more understanding about APDU structure, see Demo Applet APDU specification.

**Some restriction:**

* AUTH key must be 16 bytes.
* HMAC key must not be more than 64 bytes.
* For Wrap/Unwrap service, DES key must be 16 or 24 bytes.
* For Wrap/Unwrap, if NOPAD algorithm is used (such as: ALG\_DES\_CBC\_NOPAD), then the input message must be explicitly padded according to multiple of the key length. For example: if 16 bytes DES key with NOPAD algorithm (ALG\_DES\_CBC\_NOPAD) is used, then the plain text for wrap/unwrap must be multiple of 16 bytes (explicit padding is needed). Otherwise 6F00 status word will be thrown.
* RSA public key only for sign verification.
* EC public key is for both sign verification and key agreement.

# Random Number Generation Service

This service is for generating random number of given size. To generate random number, the Get Challenge service has two ways:

1. Generate random number with seeding
2. Generate random number without seeding.

In generating random number with seeding the seed value is required and its minimum size is 8 (eight) bytes.

**Supported algorithms:**

* ALG\_SECURE\_RANDOM
* ALG\_PSEUDO\_RANDOM

**Steps to follow for Random Number Generation Service:**

1. Perform Authentication (see section: Authenticate Service).
2. Use GET CHALLANGE command as specified in “Demo Applet APDU Specification” document for Random Number Generation Service.

Sample APDU for generating random number with 8 bytes seeding:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| CLA | INS | P1 | P2 | LC | Data | LE |
| 00 | 84 | 00 | 00 | 0A | 08 20 eight bytes seed value | -- |

# Key Pair Generation Service

Key Pair Generation service generates two types of key pair:

1. RSA key pair.
2. EC key pair.

## RSA key pair:

Key Pair Generation service only generates RSA-2048 key pair.

## EC key pair:

Demo Applet supports all the NIST and FIPS approved EC key pair generation:

1. EC P-224.
2. EC P-256.
3. EC P-384.
4. EC P-521.

This Applet preserves the generated key pair into its internal key objects for further use until the Destroy (See section: **Error! Reference source not found.**) service is called or a new key pair is generated! When a new key pair is generated the previous key pair is rewrite with the new key pair. After the key pair is generated, only the public key can be retrieved by GET DATA APDU command.

**Steps to follow for Key Pair Generation Service:**

1. Perform Authentication (see section: Authenticate Service).
2. Use Key Pair Generation command as specified in “Demo Applet APDU Specification” document for Key Pair Generation service.

Sample APDU for generating EC-521 key pair:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| CLA | INS | P1 | P2 | LC | Data | LE |
| 00 | 46 | 00 | 00 | 01 | 15 | -- |

# GET Data Service

GET DATA service is to retrieve the public key from the generated key pair (See section: Key Pair Generation). Only the public key is retrievable from the key pair.

GET DATA service retrieve public key in two ways:

1. To retrieve RSA public key, one APDU command’s response will return the public key modulus (1024 or 2048bits). And other APDU command’s response will return the public key exponent.
2. To retrieve EC public key one APDU command’s response will return the EC public key.

**Public key format:**

* APDU command to retrieve modulus of RSA public key returns the modulus value of the key in plain text. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).
* APDU command to retrieve exponent of RSA public key returns the public exponent value of the key in plain text. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).
* APDU command to retrieve EC public key returns the point of the EC curve comprising the public key in plain text form. The point is represented as an octet string in compressed or uncompressed forms as per ANSI X9.62. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

**Supported keys**:

* RSA public key.
* EC public key.

**Steps to follow for GET Data service:**

1. Perform Authentication (see section: **Error! Reference source not found.**).
2. Use GET DATA command as specified in “Demo Applet APDU Specification” document for getting the public key.

Sample APDU for getting the EC public key:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| CLA | INS | P1 | P2 | LC | Data | LE |
| 00 | CA | 01 | 60 | -- | -- | -- |

# Manage Security Environment (MSE) Service

To perform any security or cryptographic operation appropriate algorithm is needed. Manage Security Environment (MSE) Service will initialize the corresponding Security Environment (SE) with this algorithm. Then the cryptographic operation will be done based on the algorithm settled in SE.

**Supported algorithms:** See Demo Applet APDU specification’s MSE part for supported algorithm’s list.

MSE has two main parts:

1. Restore SE
2. SET SE

## Restore

In Restore process, corresponding SE will be set with some predefined algorithms.

* Security Environment for *HASH(Message Digest Service)* operation will be restored with algorithm *ALG\_SHA\_256*
* Security Environment for *CONFIDENTIALITY (Key Wrap Service)* operation will be restored with algorithm *ALG\_AES\_BLOCK\_128\_CBC\_NOPAD*
* Security Environment for *DS (Digital Signature and Message Authentication service)* operation will be restored with algorithm *ALG\_AES\_MAC\_128\_NOPAD*.

In a session, the cards SE must be restored first at least one time. Without restoring the SE, one cannot SET it!

**Steps to follow for Restore service:**

1. Perform Authentication (see section: **Error! Reference source not found.**).
2. Use MSE-RESTORE command as specified in “Demo Applet APDU Specification” document for restoring the SE.

Sample APDU for restoring SE:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| CLA | INS | P1 | P2 | LC | Data | LE |
| 00 | 22 | F3 | 00 | -- | -- | -- |

## SET

Applet user can SET SE with any preferred algorithm for appropriate security operation. But SE must be restored once in a session before SET. If the SE is not resorted at least once in a session, it cannot be SET!

Suppose, performing Wrap operation, SE need to be SET for Confidentiality operation with appropriate wrapping algorithm. See Demo Applet APDU specification for wrapping algorithm list.

Whenever a security operation is going to perform, first, the SE for that operation is checked for proper algorithm. According to the algorithm found in SE, appropriate Key container is searched to initialize the Key.

For example, if SE is set with algorithm ALG\_DES\_CBC\_PKCS5 for Confidentiality (wrap/unwrap) operation, then, to initialize appropriate Key for wrap/unwrap operation, the DEM-KEY-WRAP key containers will be searched for key data. If DEM-KEY-WRAP key containers are not initialized with **DES** **key**, then Key initialization will be failed.

See section Update Demo Keys Service for which Key containers will be searched for Key data to perform what kind of security operation.

So, if applet user updates DEM-3P-PUB key containers with RSA-1024 public key to perform Signature verification but do not SET SE with appropriate algorithm for DS operation then unexpected behavior may be found!

**Steps to follow for SET SE service:**

1. Perform Authentication (See section: **Error! Reference source not found.**).
2. Perform Restore SE (See section: Restore SE service) at least once in every session.
3. Use MSE-SET command as specified in “Demo Applet APDU Specification” document for setting SE.

Sample APDU for setting SE with ALG\_HMAC\_SHA1 for DS operation:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| CLA | INS | P1 | P2 | LC | Data | LE |
| 00 | 22 | F1 | B6 | 03 | 8001 0C | -- |

# Digital Signature Service

Digital Signature (DS) service mainly does two things:

1. Signature generation.
2. Signature verification.

## Signature generation

This service generates signature from a plain text.

First the applet user Restore or SET the SE for DS operation with appropriate algorithm. Then, the Key for Signature generation will be initialized internally according to the algorithm settled in SE.

If the algorithm is for RSA or EC then the Signature Key will be initialized from the corresponding RSA or EC private key, generated in Key Pair Generation service (See section: ). So Key Pair Generation service must be performed before DS service. Otherwise unexpected behavior may be found.

The highest length of the data to be signed is 255 bytes.

**Supported keys**:

* RSA-2048 key pair.
* All EC key pair approved by NIST and FIPS .

**Supported Algorithms**:

* ALG\_RSA\_SHA\_224\_PKCS1
* ALG\_RSA\_SHA\_224\_PKCS1\_PSS
* ALG\_RSA\_SHA\_256\_PKCS1
* ALG\_RSA\_SHA\_256\_PKCS1\_PSS
* ALG\_RSA\_SHA\_384\_PKCS1
* ALG\_RSA\_SHA\_384\_PKCS1\_PSS
* ALG\_RSA\_SHA\_512\_PKCS1
* ALG\_RSA\_SHA\_512\_PKCS1\_PSS
* ALG\_ECDSA\_SHA
* ALG\_ECDSA\_SHA\_224
* ALG\_ECDSA\_SHA\_256
* ALG\_ECDSA\_SHA\_384
* ALG\_ECDSA\_SHA\_512

**Steps to follow for Signature generation service:**

1. Perform Authentication Service.
2. Perform SE Service.
3. Perform (At least once as key pair is preserved internally).
4. Use Perform Security Operation (PSO) command as specified in “Demo Applet APDU Specification” document for Signature generation.

Sample APDU for generating DS:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| CLA | INS | P1 | P2 | LC | Data | LE |
| 00 | 2A | 9E | 9A | Data length | Data to be signed | -- |

## Signature verification

This service verifies a signature data.

First the applet user Restore or SET the SE for DS operation with appropriate algorithm. Then, the Key for Signature generation will be initialized internally according to the algorithm settled in SE.

If the algorithm is for RSA or EC then the DEM-3P-PUB key containers will be searched internally for key data to initialize the Signature Key with corresponding key types: RSA public key or EC public key.

So **Error! Reference source not found.** must be performed before DS service. Otherwise unexpected behavior may be found.

To verify a signed data, multiple APDU command is needed. Demo applet do not support update operation in sign verification. The highest length of signed data can be verified is 508 bytes. And the highest length of the data to be verified is 254 bytes.

**If the signed data is no more than 254 bytes then:**

* First APDU command will send the signed data started with a tag “20”.
* Second APDU command will send the data to be verified started with a tag “10”.

**If the signed data length is larger than 254 bytes then:**

Two APDU command will needed to send the signed data and third APDU command will send the data to be verified.

* First APDU command will send the first part (254 bytes without tag) of signed data started with a tag “20”.
* Second APDU command will send the last part (254 bytes without tag) of signed data started with a tag “40”.
* And the third APDU command will send the data to be verified, started with a tag “10”.

User cannot send last part of signed data before the first part of signed data is already sent. Moreover, the sign verification process starts when the data to be verified is sent! So user cannot send the data to be verified before the first part of signed data is already sent.

**Supported keys**:

* RSA public keys (both 1024 and 2048 bits).
* EC public key (all EC public keys).

**Supported Algorithms**:

* ALG\_RSA\_SHA\_ISO9796
* ALG\_RSA\_SHA\_PKCS1
* ALG\_RSA\_SHA\_PKCS1\_PSS
* ALG\_RSA\_SHA\_RFC2409
* ALG\_RSA\_SHA\_224\_PKCS1
* ALG\_RSA\_SHA\_224\_PKCS1\_PSS
* ALG\_RSA\_SHA\_256\_PKCS1
* ALG\_RSA\_SHA\_256\_PKCS1\_PSS
* ALG\_RSA\_SHA\_384\_PKCS1
* ALG\_RSA\_SHA\_384\_PKCS1\_PSS
* ALG\_RSA\_SHA\_512\_PKCS1
* ALG\_RSA\_SHA\_512\_PKCS1\_PSS
* ALG\_ECDSA\_SHA
* ALG\_ECDSA\_SHA\_224
* ALG\_ECDSA\_SHA\_256
* ALG\_ECDSA\_SHA\_384
* ALG\_ECDSA\_SHA\_512

**Steps to follow for Signature verification service:**

1. Perform Authentication Service.
2. Perform SE Service.
3. Perform 3.4 Update DEM-3P-PUB key containers service.
4. Use Perform Security Operation (PSO) command as specified in “Demo Applet APDU Specification” document for Signature verification.

Sample APDU for verify DS:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| CLA | INS | P1 | P2 | LC | Data | LE |
| 00 | 2A | 00 | A8 | Data length | 20 + signed data | -- |
| 00 | 2A | 00 | A8 | Data length | 10 + data to be verified | -- |

# Message Authentication Service

Message Authentication (MAC) service mainly does two things:

1. MAC generation.
2. MAC verification.

## MAC generation

This service generates MAC from a plain text

First the applet user Restore or SET the SE for DS operation with appropriate algorithm. Then, the Key for MAC generation will be initialized internally according to the algorithm in SE.

If the algorithm is for DES, AES or HMAC then the DEM-MAC key containers will be searched internally for key data with corresponding key types like DES, AES or HMAC, to initialize the MAC Key. So **Error! Reference source not found.** must be performed before DS service.

**Supported keys**:

* DES key (16 or 24 bytes).
* AES-128 key (16 bytes).
* HMAC key (Not more than 64 bytes).

**Supported Algorithms**:

* ALG\_AES\_MAC\_128\_NOPAD
* ALG\_DES\_MAC4\_ISO9797\_1\_M2\_ALG3
* ALG\_DES\_MAC4\_ISO9797\_M1
* ALG\_DES\_MAC4\_ISO9797\_M2
* ALG\_DES\_MAC4\_NOPAD
* ALG\_DES\_MAC4\_PKCS5
* ALG\_DES\_MAC8\_ISO9797\_1\_M2\_ALG3
* ALG\_DES\_MAC8\_ISO9797\_M1
* ALG\_DES\_MAC8\_ISO9797\_M2
* ALG\_DES\_MAC8\_NOPAD
* ALG\_DES\_MAC8\_PKCS5
* ALG\_HMAC\_SHA1
* ALG\_HMAC\_SHA\_256
* ALG\_HMAC\_SHA\_384
* ALG\_HMAC\_SHA\_512

**Steps to follow for MAC generation service:**

1. Perform Authentication Service.
2. Perform SE Service.
3. Perform 3.2 Update DEM-MAC Key Container service.
4. Use Perform Security Operation (PSO) command as specified in “Demo Applet APDU Specification” document for MAC generation.

Sample APDU for generating MAC:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| CLA | INS | P1 | P2 | LC | Data | LE |
| 00 | 2A | 9E | 9A | Data length | Data for generating MAC | -- |

## MAC verification

This service verifies MAC data.

First the applet user Restore or SET the SE for DS operation with appropriate algorithm. Then, the Key for MAC generation will be initialized internally according to the algorithm in SE.

If the algorithm is for DES, AES or HMAC then the DEM-MAC key containers will be searched internally for key data with corresponding key types like DES, AES or HMAC, to initialize the MAC Key. So **Error! Reference source not found.** must be performed before DS service.

To verify MAC, multiple APDU command is needed. Demo applet do not support update operation in MAC verification. The highest length of MAC to be verified is 508 bytes. And the highest length of the data to be verified is 254 bytes.

**If the MAC is no more than 254 bytes then:**

* First APDU command will send the MAC with a tag “20”.
* Second APDU command will send the data to be verified started with a tag “10”.

**If the MAC data length is larger than 254 bytes then:**

Two APDU command will needed to send the MAC and third APDU command will send the data to be verified.

* First APDU command will send the first part (254 bytes without tag) of MAC started with a tag “20”.
* Second APDU command will send the last part (254 bytes without tag) of MAC started with a tag “40”.
* And the third APDU command will send the data to be verified, started with a tag “10”.

User cannot send last part of MAC before the first part of MAC is already sent. Moreover, the MAC verification process starts when the data to be verified is sent! So user cannot send the data to be verified before the first part of MAC is already sent.

**Supported keys**:

* DES key (16 or 24 bytes).
* AES-128 key (16 bytes).
* HMAC key (Not more than 64 bytes).

**Supported Algorithms**: Same as MAC generation.

**Steps to follow for MAC verification service:**

1. Perform Authentication Service.
2. Perform SE Service.
3. Perform 3.2 Update DEM-MAC Key Container service.
4. Use Perform Security Operation (PSO) command as specified in “Demo Applet APDU Specification” document for MAC verification.

Sample APDU for verify MAC:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| CLA | INS | P1 | P2 | LC | Data | LE |
| 00 | 2A | 00 | A8 | Data length | 20 + MAC | -- |
| 00 | 2A | 00 | A8 | Data length | 10 + data to be verified | -- |

# Message Digest Service

This service is for generating message digest with preferred algorithm.

First the applet user Restore or SET the SE for HASH operation with appropriate algorithm. Then an APDU command with maximum 255 bytes data for making digest will be send.

**Supported Algorithms**:

* ALG\_SHA
* ALG\_SHA\_256
* ALG\_SHA\_384
* ALG\_SHA\_512
* ALG\_SHA\_224

**Steps to follow for Message Digest Service:**

1. Perform Authentication Service.
2. Perform SE Service.
3. Use Perform Security Operation (PSO) command as specified in “Demo Applet APDU Specification” document for Message Digest Service.

Sample APDU for Message Digest:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| CLA | INS | P1 | P2 | LC | Data | LE |
| 00 | 2A | 90 | 80 | Data length | Data for message digest | -- |

Response: Digest.

# Key Wrap Service

This service is mainly encrypts or decrypts a Key or plain text. So the service is divided into two parts:

1. Wrap.
2. Unwrap.

## Wrap:

Wrapping means encrypting plain text. The plain text may be a key or plain data!

To perform this service properly, applet user must Restore or SET SE for Confidentiality with appropriate algorithm first.

According to the algorithm settled in SE, the DEM-KEY-WRAP key containers will be searched for key data to initialize the Wrap Key. So **Error! Reference source not found.** must be performed before Wrap service.

Sample APDU for Wrap:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| CLA | INS | P1 | P2 | LC | Data | LE |
| 00 | 2A | 86 | 80 | Data length | Plain data | -- |

Response: Encrypted data.

## Unwrap:

Unwrapping means decrypt text (plain data/text). The plain text may be an encrypted key or plain data!

To perform this service properly, applet user must Restore or SET SE for Confidentiality with appropriate algorithm first.

According to the algorithm settled in SE, the DEM-KEY-WRAP key containers will be searched for key data to initialize the Unwrap Key. So **Error! Reference source not found.** must be performed before Unwrap service.

Sample APDU for Unwrap:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| CLA | INS | P1 | P2 | LC | Data | LE |
| 00 | 2A | 80 | 86 | Data length | Plain/encrypted data | -- |

Response: Decrypted data.

# Key Agreement Service

Demo Applet’s Key Agreement service uses all EC key pairs suggest by NIST and FIPS.

Through Key Agreement service two or more parties, like off-card and on-card, can agree on a Key/Secret in such a way that both influence the outcome.

To generate the Secret, Key Agreement service needs the EC curves name, like EC-384 or EC-521 and EC public key in uncompressed forms as per ANSI X9.62 by an APDU command.

Key Agreement service generates an EC Key Pair with the EC curve name and use the private key and public key from APDU to generate the Secret.

In response the Key Agreement service returns the Public key of the key pair it generates!

Sample APDU for Key Agreement with EC-224:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| CLA | INS | P1 | P2 | LC | Data | LE |
| 00 | 86 | 12 | 00 | Data length | Public key of EC-224 curve | -- |

# Destroy Service

This service will initialize all the Key Containers with zeros! And all of the Demo Applet’s internal key objects that preserve the keys/key pairs for further use will clear!

Hence no operation can be done using those objects or key containers.

Sample APDU for Destroy:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| CLA | INS | P1 | P2 | LC | Data | LE |
| 00 | 30 | 00 | 00 | -- | -- | -- |