**Toolkit Framework**

**(Release 1)**

**Contents**

[1. **Proprietary File Architecture** 3](#_Toc119939513)

[2. **DFPF (Proprietary File)** 3](#_Toc119939514)

[**2.1.** **EFRFMP (RFM Parameter)** 3](#_Toc119939515)

[**2.2.** **EFTP (Terminal Profile):** 5](#_Toc119939516)

[**2.3.** **Create File (EF):** 5](#_Toc119939517)

[**2.4.** **EF (Key management).** 6](#_Toc119939518)

[**2.5. EF (Key Values):** 7](#_Toc119939519)

[**2.6. EF (Secured Message Data):** 8](#_Toc119939519)

[3. Algorithms Implementation 9](#_Toc119939520)

[**Methods: -** 9](#_Toc119939521)

### **­Proprietary File Architecture**

### **DFPF (Proprietary File)**

|  |  |  |
| --- | --- | --- |
| **Identifier : '7F49''** | **Structure : DF** | **Mandatory** |
| Access Conditions:  CREATE ADM  DELETE ADM  DEACTIVATE ADM  ACTIVATE ADM | | |

#### **2.1. EFRFMP (RFM Parameter)**

EFRFMPI is a linear fixed file under the DF(7F49) and is under the responsibility of the issuer.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Identifier : '6F16'' | | Structure : linear fixed | | | Mandatory | |
| Record length: 29 bytes | | | Update activity: low | | | |
| Access Conditions:  READ ADM  UPDATE ADM  DEACTIVATE ADM  ACTIVATE ADM | | | | | | |
| **Bytes** | **Description** | | | **M/O** | | **Length** |
| 1 to 3 | Access domain | | | M | | 3 bytes |
| 4 | Minimum Security Level (MSL) | | | M | | 1 byte |
| 5 | Length of AID (X) | | | M | | 1 byte |
| 6 to (5 + X) | AID | | | M | | X bytes |
| 22 to 26 | CNTR | | | M | | 5 bytes |
| 27 to 29 | TAR | | | M | | 3 bytes |

The EF consists of one or more records, with each record able to hold one entry corresponding to given TAR (application).

Each record contains the AID of ADF corresponding to given TAR.

**Access domain**

The Access Domain is a parameter used to define the access rights granted to an application allowing it to perform operations on UICC files specified in ETSI TS 102 221. In table 1 defined Coding of Access domain.

**Table 1: Coding of the Access Domain**

|  |  |
| --- | --- |
| **Value** | **Description** |
| ‘FFFFFF’ | Full access to the File System |
| ‘XXXXXX’(Note1) | UICC access mechanism |
| ‘000000’ | No access to the File System |

Note1: The UICC access mechanism shall be coded as follows:

Diagram

Description automatically generated

These access rights shall be checked against SE ID 01 access rules as defined in ETSI TS 102 221.

**Minimum Security Level (MSL)**

The Minimum-Security Level is the Minimum SPI1 which shall use the same coding as the first octet of the SPI of a command packet (see clause 5.1.1 of ETSI TS 102 225).

The first octet of the SPI field in the incoming message Command Packet (SPI1) shall be checked against the Minimum-Security Level Data (MSLD) byte by the receiving entity according to the following rules:

* if SPI1.b2b1 is equal to or greater than MSLD.b2b1.
* if SPI1.b3 is equal to or greater than MSLD.b3; and
* if SPI1.b5b4 is equal to or greater than MSLD.b5b4.

then the Message Security Level is sufficient, and the check is successful, otherwise the check is failed i.e. “Insufficient Security Level”.

**AID(Application Identifier)**

A selectable application, represented in the UICC by the AID, shall be referenced by a DF name coded on 1 byte to 16 bytes. Each name shall be unique within a UICC.

NOTE 1: The TAR of an ADF RFM application shall be linked to the AID of the application to which the ADF belongs.

NOTE 2: The Length of AID (X) byte shall contain the length of linked AID of the application to which the ADF belongs, otherwise it shall be fixed to 0.

**CNTR**

Mechanism or data field used for keeping track of a message sequence.

NOTE : This could be realized as a sequence oriented, maintaining a level of synchronization between the Sending Entity and the Receiving Entity.

**TAR**

The Toolkit Application Reference (TAR) is used to uniquely identify an RFM. The TAR is defined and coded according to ETSI TS 101 220.

#### **2.2. EFTP (Terminal Profile):**

This EF describe the CAT facilities that are supported by the terminal.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Identifier: ‘6F17’ | | Structure: transparent | | Mandatory | | |
| SFI : Not Supporter | | |  | | | |
| File size: 35 bytes | | | Update Activity : low | | | |
| Access Conditions :  READ NEV  UPDATE NEV  DEACTIVATE NEV  ACTIVATE NEV | | | | | | |
| **Bytes** | **Description** | | | | **M/O** | **Length** |
| 1 to 35 | Profile | | | | M | 35 |

NOTE: Only the OS can use this EF internally and it will not update externally.

• Profile:

Contents: According to Recommendation ETSI TS 102 223 clause 5.2.

Purpose: Store Profile.

#### **2.3. Create File (EF):**

CREATE is formatted as described in ETSI TS 102 222 (clause 6.3). Then the content of OTA file information will be formatted as described below:

This is the last TLV in CREATE command and other TLVs occurring after this shall be ignored

**Tag ‘CC’: OTA File Information within the proprietary TLV (Tag ‘A5’).**

|  |  |  |  |
| --- | --- | --- | --- |
| Bytes | Description | Value | Length |
| 1 | Tag | ‘CC’ | 1 |
| 2 | Length | 1 | 1 |
| 3 | Coding of OTA file Information (table 1) |  | 1 |

**Table 1: coding of OTA file information.**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| b8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | Meaning |
| - | - | - | - | - | - | - | 0 | RFM Not updatable |
| - | - | - | - | - | - | - | 1 | RFM updatable |
| X | X | X | X | X | X | X | - | RFU |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Identifier: '6F18' | | Structure: linear fixed | | | Mandatory | |
| Record length: 5 bytes | | | Update activity: low | | | |
| Access Conditions:  READ ADM  UPDATE ADM  DEACTIVATE ADM  ACTIVATE ADM | | | | | | |
| **Bytes** | **Description** | | | **M/O** | | **Length** |
| 1 | KEY VALUE SIZE | | | M | | 1 byte |
| 2 | BLOCK OFFSET | | | M | | 1 byte |
| 3 | ALGORITHM (DES, AES) | | | M | | 1 byte |
| 4 | KEY ID (KIC, KID) | | | M | | 1 byte |
| 5 | KVN | | | M | | 1 byte |

#### **2.4. EF (Key management).**

This EF describes the Keys configuration of keysets used in SMS-PP download (ref. ETSI 102 225).

It is located under ’7F49’.

**- KVN**

Contents: Key Version Number: Supported ’01-0F’

**- KEY ID**

Contents: Key Identifier, Each KVN will have multiple Key Identifiers. Current implementation supports only two:

**‘00’**: Ciphering Key Identifier (KIC)

**‘01’**: Integrity Key Identifier (KID)

**- ALGORITHM**

Contents: Key type, algorithm usage of given key.

Supported values are:

Triple DES: ‘10’

AES: ‘11'

**- KEY VALUE SIZE**

Contents: It stores the size of key value. It must be a multiple of 8

**- BLOCK OFFSET**

It provides the offset from where key value starts in ‘6F19’. The actual offset to EF (Key Values) is calculated by multiplying Block Offset with 8

#### **2.5. EF (Key Values):**

This EF contains the Key values. It is located under ’7F49’.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Identifier: '6F19' | | Structure: Transparent | | | Mandatory | |
| FILE SIZE: X Bytes | | | Update activity: low | | | |
| Access Conditions:  READ ADM  UPDATE ADM  DEACTIVATE ADM  ACTIVATE ADM | | | | | | |
| **Bytes** | **Description** | | | **M/O** | | **Length** |
| 1 to X | KEY VALUES | | | M | | X byte |

File Size (value of X) will be the sum of size of all the Key Values stored in this EF.

**Example:**

Let’s say if we want to add two keysets of KVN ‘01’ and KVN ’02’ of type 3DES and AES respectively, Content of above two files are following:

6F18:

Record 1: 1800100002

Record 2: 1803100102

Record 3: 1006110001

Record 4: 1008110101

6F19:

<1111….1111> <2222….2222> <3333….3333> <4444….44444>

|------24------| |------24------| |------16------| |------16------|

The block size of previous key value should be added to the block offset.

* There are 4 key values stored in ‘6F19’, so there will be 4 records of record length = 05 bytes in ‘6F18’
* Then the file size of ‘6F18’ will be (4\*05) 20 bytes. And the file size of ‘6F19’ will be (24+24+16+16)

80 bytes.

#### **2.6. EFSMD (Secured Message Data):**

This EF stores Secured Message Data temporarily. No. of concatenated messages are configurable as per requirements and on this basis the file size of this EF depends on the configuration.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Identifier: ‘6F21’ | | Structure: transparent | | | Mandatory | | |
| SFI : Not Supporter | | |  | | | | |
| File size: : (150\*(X+Y))+4 bytes | | | Update Activity : high | | | | |
| Access Conditions :  READ ADM  UPDATE ADM  DEACTIVATE ADM  ACTIVATE ADM | | | | | | | |
| **Bytes** | **Description** | | | **Value** | | **M/O** | **Length** |
| 1 | No. of concatenated Packets supported | | | X | | M | 1 |
| 2 | No. of concatenated Response supported | | | Y | | M | 1 |
| 3-4 | Length of Secured message | | | - | | M | 2 |
| As per X | Secured Message | | | - | | M | 150\*X |
| As per Y | Response Message | | | - | | M | 150\*Y |

### **Algorithms Implementation**

Methods: -

|  |  |
| --- | --- |
| **Modifier and Type** | **Method and Description** |
| void | **init(**theKey, theMode, bArray, bOff, bLen)  Initializes the Cipher object with the appropriate Key and algorithm specific parameters. |
| short | **update**(inBuff, inOffset, inLength, outBuff, outOffset)  Generates encrypted/decrypted output from input data. |
| short | **doFinal**(inBuff, inOffset, inLength, outBuff, outOffset)  Generates encrypted/decrypted output from all/last input data. |

**init**: -

***init(****theKey, theMode, bArray, bOff, bLen);*

init() must be used to update the Cipher object with a new key. If the Key object is modified after invoking the init() method, the behavior of the update() and doFinal() methods is unspecified.

The Key is checked for consistency with the Cipher algorithm. For example, the key type must be matched

**Parameters:**

theKey - the key object to use for encrypting or decrypting.

theMode - one of MODE\_DECRYPT or MODE\_ENCRYPT

bArray - byte array containing algorithm specific initialization info

bOff - offset within bArray where the algorithm specific data begins

bLen - byte length of algorithm specific parameter data

**update: -**

***short update****(inBuff, inOffset, inLength, outBuff, outOffset);*

Generates encrypted/decrypted output from input data. This method is intended for multiple-part encryption/decryption operations.

This method should only be used if all the input data required for the cipher is not available in one byte array. If all the input data required for the cipher is located in a single byte array, use of the doFinal() method to process all of the input data is recommended. The doFinal() method must be invoked to complete processing of any remaining input data buffered by one or more calls to the update() method.

**Parameters:**

* inBuff - the input buffer of data to be encrypted/decrypted
* inOffset - the offset into the input buffer at which to begin encryption/decryption
* inLength - the byte length to be encrypted/decrypted
* outBuff - the output buffer, may be the same as the input buffer
* outOffset - the offset into the output buffer where the resulting ciphertext/plaintext begins

**Returns:**

number of bytes output in outBuff

**doFinal: -**

***short doFinal****(inBuff, inOffset, inLength, outBuff, outOffset);*

Generates encrypted/decrypted output from all/last input data. This method must be invoked to complete a cipher operation. This method processes any remaining input data buffered by one or more calls to the update() method as well as input data supplied in the inBuff parameter.

**Parameters:**

* inBuff - the input buffer of data to be encrypted/decrypted
* inOffset - the offset into the input buffer at which to begin encryption/decryption
* inLength - the byte length to be encrypted/decrypted
* outBuff - the output buffer, may be the same as the input buffer
* outOffset - the offset into the output buffer where the resulting output data begins

**Returns:**

number of bytes output in outBuff

**Throws:**

Exception with the following reason codes:

* UNINITIALIZED\_KEY if key not initialized.
* INVALID\_INIT if this Cipher object is not initialized.
* ILLEGAL\_USE if one of the following conditions is met:
  + This Cipher algorithm does not pad the message and the message is not block aligned.
  + This Cipher algorithm does not pad the message and no input data has been provided in inBuff or via the update() method.
  + The input message length is not supported or the message value is greater than or equal to the modulus.
  + The decrypted data is not bounded by appropriate padding bytes.