

AN14215

MIFARE DUOX features and hints

Rev. 1.0 — 13 November 2024
975010

Application note
CONFIDENTIAL

Document information

Information	Content
Keywords	MIFARE DUOX, ISO/IEC 14443, Delegated Application Management, Key Management, Transaction MAC, Virtual Card Architecture, Proximity Check, Secure Dynamic Messaging, NFC, ECC, Signature, Authentication
Abstract	This application note addresses the functionality and the extended features of MIFARE DUOX and gives hints for usage and implementation.



1 Introduction

In this document the advanced and new features of MIFARE DUOX will be presented and explained in detail. Useful implementation hints will be given in order to make the use and integration of MIFARE DUOX easier.

Most of the cryptographic functionalities will be discussed and explained with the help of examples in order to facilitate the implementation.

It is recommended to use a Secure Application Module (SAM) Ref. [3] together with a MIFARE DUOX reader / terminal to establish a higher level of security in case of secure key storage needed also on the reader side.

MIFARE DUOX is Common Criteria EAL6+ security certified ICs, which fully complies with the requirements for fast and highly secure data transmission and flexible application management. As the first product in the MIFARE family, MIFARE DUOX provides asymmetric cryptography functions and authentication based on ECC-256 public and private keys.

1.1 About this document

This document addresses developers and people who already have general know-how of the MIFARE DUOX and its command set.

Please note that this document does not cover the general working principle of the MIFARE DUOX. Please read Ref. [1] in order to get the full overview and description of the MIFARE DUOX.

This application note is a supplementary document for implementations using the MIFARE DUOX. Should there be any confusion, please check the MIFARE DUOX datasheet Ref. [1]. The best use of this application note will be achieved by reading the mentioned datasheet in advance.

Note: This application note does not replace any of the relevant functional specifications, datasheets or design guides.

1.2 MIFARE DUOX Support Package

Like all other MIFARE products, MIFARE DUOX comes together with a product support package (PSP). It is advised to collect the required items from the product support package before starting development as they contain useful information, detailed command description, supporting examples as well as assisting software. The following list reflects the available support items:

1.2.1 Documents

Datasheet: MIFARE DUOX datasheet: DS9744XX

Application Note: MIFARE DUOX Features and Hints: AN14215

Application Note: MIFARE DUOX Antenna Design Guide: AN14216

Application Note: MIFARE DUOX Quick-Start Guide: AN14217

Application Note: Feature and Functionality Comparison between MIFARE DUOX and other MIFARE DESFire products: AN14218

Application Note: MIFARE DUOX Customization and Trust Provisioning Process: AN14222

1.2.2 Software

- RFID Discover – a software tool to evaluate MIFARE DUOX and send commands to the card, can be ordered via your NXP contact.

- SW1866 RFIDDiscover – Contains the full command set of MIFARE DUOX and should be used in order to make use of the full card functionality. It can be obtained via NXP secure files.
- SW5434 Card Test Framework – Contains the full command set of MIFARE DUOX and offers the possibility of scripting those commands into full transactions. This can be useful for development, testing and POC.

1.2.3 Hardware

- MIFARE DUOX samples can be ordered via your NXP contact.

1.2.4 Trainings

- MIFARE DUOX technical trainings and hands-on workshops are provided in different regions all over the world.
- You can find an overview of all offered trainings on the NXP website

Get in touch with your local NXP contact person for any support and unique inquiry you need.

1.3 Organization of this document

This document describes the relevant information for designing and implementing a MIFARE DUOX system starting with an Introduction. Chapter 3 of this document shows ISO/IEC 14443 standard, chapter 4 highlights ISO/IEC 7816 whilst chapters 5-8 describe secure messaging, key management and different crypto options in detail, and finally the last chapters provide some commands and file management details.

It is assumed in this application note, that the reader has already read the Functional Specification of MIFARE DUOX Ref. [1] and has adequate knowledge about ISO/IEC 14443 Ref. [2].

The following symbols have been used in this document to mention the operations for the examples:

- = Preparation of data by the reader/host
- > Data sent from PCD to PICC
- < Data sent from PICC to PCD

All the numerical examples are just examples, describing the usage of commands and provide reference values to verify any implementation.

If the command parameters consist of multiple bytes, they are represented least significant byte (LSB) first in the given examples.

Note: Any data, value and cryptogram is expressed here in hex string format if not otherwise specified.

2 ISO/IEC 14443 and MIFARE DUOX

MIFARE DUOX supports ISO/IEC 14443-3 and ISO/IEC 14443-4 of PICC Type A. It allows configuration of several features at ISO14443 layer to make it compliant to broader implementations.

2.1 ISO/IEC 14443-3

2.1.1 SAK

MIFARE DUOX presents 7 byte UID as default at the ISO14443-3 anticollision and activation. In the case of 7 byte UID, the default SAK at cascade level 1 (CL1) is set to 0x24 and to 0x20 at cascade level 2 (CL2).

By using the Cmd.SetConfiguration with option 0x03, SAK at CL1 and SAK at CL2 can be updated to any given values.

An example of using Cmd.SetConfiguration to set SAK values is elaborated in Ref. [1].

2.1.2 Random ID

MIFARE DUOX can be configured to present a Random ID at the ISO14443-3 anticollision procedure. For Random ID, the option UID0 is set to 0x08 according to ISO/IEC 14443-3 Ref. [2]. This Random ID option can be activated using Cmd.SetConfiguration and is irreversible, after setting it once. An example of using Cmd.SetConfiguration for setting a Random ID is shown in Ref. [1].

2.2 ISO/IEC 14443-4

2.2.1 Buffer size

MIFARE DUOX offers a frame size (FSC) up to 256 bytes. The communication buffer size FSD ($FSD \leq FSC$) can be set using the RATS command by the PCD.

If the FSD is set to more than 64 bytes there will be a difference in the communication behavior for some of the MIFARE DUOX native data management commands, depending on which command code is used. It can be chosen between the native chaining (use of additional frame with 0xAF) and the chaining according to ISO/IEC 14443-4.

Command codes which support the larger FSC and the ISO/IEC 14443-4 chaining are: 0xAD, 0x8D, 0xAB, 0x8B, 0xBA. Details on the commands can be found in Ref. [1].

The bigger frame size will be in use wherever only ISO14443-4 chaining applies, and no application level chaining is used (for example when using ISO commands such as Cmd.ISOReadBinary).

2.2.2 ATS

MIFARE DUOX offers personalization of the ATS, length up to 20 bytes. Information coded in format byte (T0) and interface bytes (TA, TB, TC) is strictly followed by the PICC.

Table 1. ATS

Byte	Default Value	Can be changed to	Comments
TL	06	01 to 14	
T0	75	b8 is fixed to 0, b7 - b5 (presence of TA, TB, and TC) can be set to any value from 0 to 7. Lower nibble (FSC) can be set to any value from 0 to 7.	Set the following bytes accordingly For using a bigger frame size of 256 bytes, the FSC needs to be set first to 0x78.
TA	77	b8 can be set to 0 or 1, b4 is fixed to 0. Other bits can be set to 0 or 1.	The set communication speed is followed by the card. Change the bits only if lower communication speed is required.
TB	81	Lower nibble (SFGI) can be set to any value from 1 to E. An FWI value smaller than 7 is not supported and can never be used.	It is not recommended to change the SFGI. FWI can be changed to tune the time-out setting.
TC	02	NAD is not supported by the PICC	Keep the default value
T1	80	Can be changed to any value and up to 15 bytes.	

2.2.3 ISO/IEC 14443-4 Chaining

ISO/IEC 14443-4 chaining, as defined in Ref. [1], is supported in the following cases:

- A PCD can always split a native command in several frames, as long as the length of the reassembled INF field remains within the supported buffer size of 60 bytes (native chaining).
- Additionally, the PICC supports ISO/IEC 14443-4 chaining to allow for larger command or response frames than 60 bytes for variants of a range of commands (ISO/IEC 14443-4 chaining).

3 ISO/IEC 7816 Support

MIFARE DUOX supports several Inter Industry Instructions (INS) as described in ISO/IEC 7816-4, to comply with some application standards e.g. APTA, RIS. The supported INS are described in Ref. [1].

Usages of the ISO/IEC 7816-4 INS require the “free directory list access without master key”. The bit 1 of PICC or application master key setting has to be set to 1. Refer to section 6.5.4.2 PICCKeySettings of Ref. [1].

3.1 ISO/IEC 7816 File Structure in MIFARE DUOX

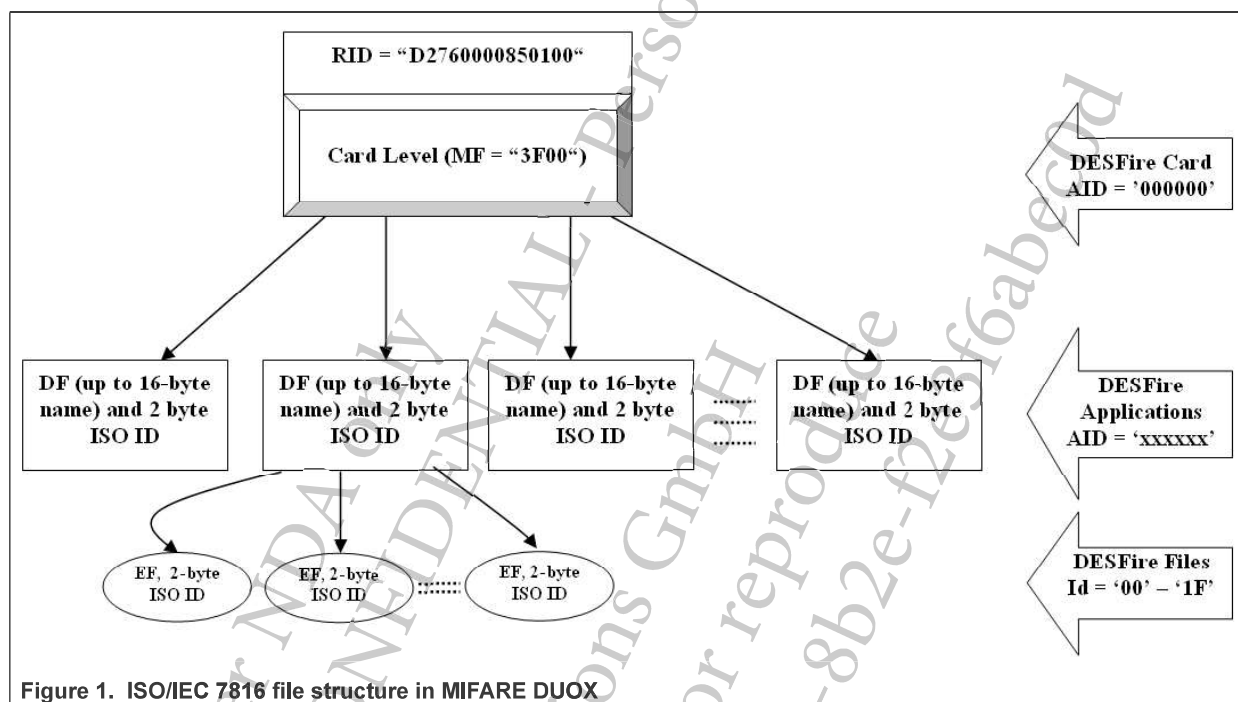


Figure 1. ISO/IEC 7816 file structure in MIFARE DUOX

3.1.1 Example: ISO/IEC 7816 File Structure in MIFARE DUOX

One application example of ISO/IEC 7816 structure is discussed in the following (DF Parameters can be seen in Table 2).

Table 2. MIFARE DUOX ISO/IEC 7816 DF Parameters

MIFARE DUOX AID	ISO/IEC7816 File ID (DF ID)	DF Names	Comments
123456	1122	010203040506	ISO/IEC 7816 file ID and DF names are defined in the <code>Cmd.CreateApplication</code>

Now within this MIFARE DUOX Application, a standard data file is created as follows.

Table 3. MIFARE DUOX ISO/IEC 7816 EF Parameters

MIFARE DUOX FID	ISO/IEC7816 File ID (EF ID)	Comments
01	0102	ISO/IEC 7816 EF ID is defined in <code>Cmd.CreateFile</code> . The b5 of <code>KeySet2</code> in the <code>Cmd.CreateApplication</code> or <code>Cmd.CreateDelgated Application</code> must be set to 1 to assign EF ID for files

Table 4. Accessing the EF in ISO/IEC 7816 Mode

Step	Command		Data Message
1 [1]	Select MIFARE DUOX RID C-APDU (<code>Cmd.IsoSelectFile</code>)	>	00A4040007D2760000850100 P1 = 04 means directory selection by DF name
2	R-APDU	<	9000
3 [2]	Select DF with DF ID 1122 C-APDU (<code>Cmd.IsoSelectFile</code>)	>	00A4000C022211 The DF ID given in <code>Cmd.CreateApplication</code> and given here in <code>Cmd.Select</code> is in different order.
4	R-APDU	<	9000
5 [3]	Select EF under the current DF (<code>Cmd.IsoSelectFile</code>)	>	00A4000C020201 The EF ID given in the <code>Cmd.CreateFile</code> and given here in <code>Cmd.Select</code> is in different order
6	R-APDU	<	9000
7 [4] [5]	Update Binary, to write in the previously selected EF (<code>Cmd.IsoUpdateBinary</code>)	>	00D60000200102030405060708091011121314151617 181920212223242526272829303132
8	R-APDU	<	9000
9 [4] [5]	Read Binary, read from the previously selected EF (<code>Cmd.IsoReadBinary</code>)	>	00B0000020
10	R-APDU	<	01020304050607080910111213141516171819202122232425262728293031329000

[1] Selecting the MIFARE DUOX RID is optional. After selection of this RID, you are still in the root directory. This selection can be made to distinguish MIFARE DUOX as the ID D2760000850100 is registered, so most probably is unique for MIFARE DUOX.

[2] The DF can be selected also with the ISO DFName. In this case for this example C-APDU = 00A4040C06010203040506.

[3] The EF selection can be skipped if the short file ID is provided in Read/Update commands.

[4] The short file ID can be provided here as well. In that case b8 of P1 is set to 1 and b5 to b1 is the short file ID. Short file ID is the lowest significant 5 bits of ISO/IEC 7816 16 bit EF ID. In this example, for FID = 0102, the short file ID = 01. So the Read Binary command for this example with short file ID the C-APDU = 00B0810020.

[5] Based on the file settings, an ISO 7816 authentication sequence may be required before accessing the EF.

3.2 Chaining in ISO/IEC 7816-4

MIFARE DUOX has the specific native chaining already known from MIFARE DESFire in addition to the ISO/IEC 14443-4 standard chaining in ISO/IEC 7816-4 type framing available.

Two examples of the ISO/IEC 14443-4 standard chaining (refer to ISO/IEC 14443-4 Ref. [2]) are shown in the following. In these examples (Section 3.2.1 and Section 3.2.2) the 2 byte CRC16 epilogue field is excluded.

3.2.1 Example: PCD uses Chaining

In this example `Cmd.IsoUpdateBinary` is used to write 99 bytes in a standard data file at offset 0. ISO/IEC 14443-4 standard chaining is used by the PCD.

Table 5. ISO/IEC 14443-4 Standard Chaining in ISO/IEC 7816 Standard Framing mode, PCD uses Chaining

Step	Command	Data Message
1	<code>Cmd.IsoUpdateBinary</code> PCB = 1A which is an I-block with chain bit set, means more frames to come, frame no = 0, CID = 01.	> 1A0100D600006301020304050607080910111213141516171819202122232425262728293031323334353637383940414243444546474849505152535455
2	PCB is R(ACK) MIFARE DUOX acknowledges receiving of frame 0.	< AA01
3	PCD send next normal I-block, with frame no 1.	> 0B015657585960616263646566676869707172737475767778798081828384858687888990919293949596979899
4	MIFARE DUOX sends normal I block with SW1SW2 = 9000	< 0B019000

3.2.2 Example: PICC uses Chaining

In this example `Cmd.IsoReadBinary` is used to read 99 bytes from a standard data file at offset 0.

Table 6. ISO/IEC 14443-4 Standard Chaining in ISO/IEC 7816 Standard Framing Mode, PICC uses Chaining

Step	Command	Data Message
1	<code>Cmd.IsoReadBinary</code> PCB = 0A which is an I-block with frame no 0, CID = 01	> 0A0100B0000063
2	Response PCB = 01 which is an I-block with the chain bit set, means more response to come.	< 1A010102030405060708091011121314151617181920212223242526272829303132333435363738394041424344454647484950515253545556575859
3	PCD sends R(ACK) with frame no = 1, which is PCB = AB	> AB01
4	Response PCB is normal I-block with frame no = 1, SW1SW2 = 9000	< 0B01606162636465666768697071727374757677787980818283848586878889909192939495969798999000
5	99 bytes application data	= 010203040506070809101112131415161718192021222324252627282930313233343536373839404142434445464748495051525354555657585960616263646566676869707172737475767778798081828384858687888990919293949596979899

3.3 Security in ISO 7816-4 Standard Mode

The security concept using ISO/IEC 7816-4 standard INS is described in Ref. [1]. Please note the limitation on data confidentiality - in this mode, user data is always transmitted plain, only in some commands a MAC (CMAC) is appended.

4 Secure Messaging

MIFARE DUOX supports the well known so called "**EV2 secure messaging**" based on AES session keys.

Note: The name "EV2" results from the fact that it was introduced with MIFARE DESFire EV2 the first time. Since MIFARE DUOX, Also AES-256 keys can be used.

The EV2 secure messaging is used regardless of the authentication method, both symmetric authentication using `Cmd.AuthenticateEV2First` and the asymmetric authentication using `Cmd.ISOGeneralAuthenticate` result in the same secure messaging using AES session keys. The asymmetric authentication is limited to a session key length of 128 bit, as this provides the same security level than ECC-256 used in the authentication itself.

When the authentication between PCD and MIFARE DUOX PICC has been performed successfully:

- A trusted communication channel is established
- The access rights related to that key are enabled
- Two session keys are generated

Note: Authentication might be required (based on file settings) to perform a certain operation or transaction, although data transfer might be done later in plain text (based on defined communication mode).

- It is only possible to be authenticated with one key at the same time.
- A new authentication invalidates the previous authentication status.
- In `Cmd.AuthenticateEV2First` (command code 0x71), authentication starts with a zero byte IV. Within the authentication procedure and after the authentication a zero byte IV is used. Within the session, the IV is recalculated for each command.
- Using `Cmd.AuthenticateEV2NonFirst` (command code 0x77), authentication starts a zero byte IV. Within the authentication procedure and after the authentication a zero byte IV is used. Within the session, the IV is recalculated for each command.
- When using `Cmd.ISOGeneralAuthenticate`, a ECDH key agreement procedure is performed, which results in 2 AES-128 session keys that are used after the authentication for secure messaging

At PICC level and application level, communication modes are defined by the command itself. At file level, communication mode is defined by the file (mode defined during the file creation).

MIFARE DUOX supports three different communication modes as shown in [Table 7](#).

Table 7. MIFARE DUOX Communication Modes

Communication Mode	Bit Representation	Explanation
<code>CommMode.Plain</code>	X0	Plain communication: No encryption is used at all.
<code>CommMode.MAC</code>	01	MACed communication: The data is transferred in plain, but an 8 byte CMAC is added to the message.
<code>CommMode.Full</code>	11	Encrypted communication: AES is used to encrypt the transferred data. An 8 byte CMAC is added to the encrypted command data.

Note:

1. If there is no active authentication, both command and response are sent in plain (command will be rejected in case an authentication is required).
2. Authentication commands as well as some other commands (e.g. key change commands, `Cmd.GetVersion`, ...) have their own secure messaging rules.

4.1 EV2 Secure Messaging

EV2 Secure Messaging gets activated after successful authentication with `Cmd.AuthenticationEV2First` (command code 0x71) or `Cmd.AuthenticationEV2NonFirst` (command code 0x77) or `Cmd.ISOGeneralAuthenticate` (command code 0x87).

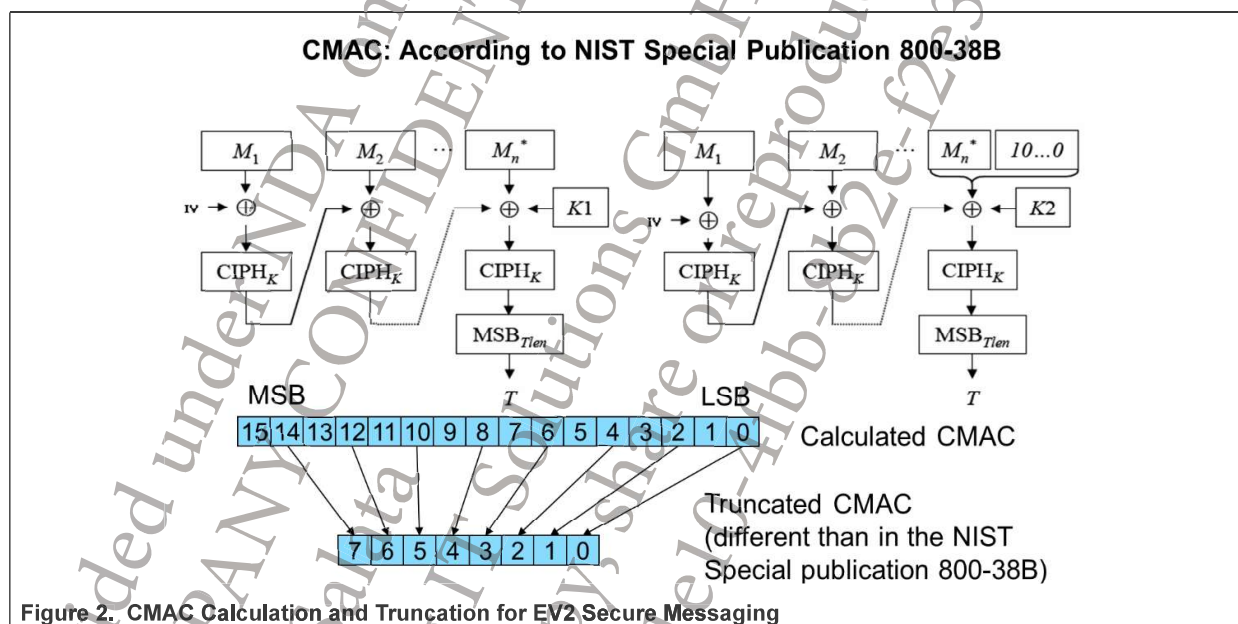
When being not authenticated, a first authentication needs to be established, using `Cmd.AuthenticationEV2First`. If an authentication was already executed using `Cmd.AuthenticationEV2First`, and EV2 Secure Messaging is already established, a subsequent authentication can be achieved by executing `Cmd.AuthenticationEV2NonFirst`. `Cmd.AuthenticationEV2NonFirst` is intended for any subsequent authentication after `Cmd.AuthenticationEV2First` in a transaction.

Asymmetric authentication using `Cmd.ISOGeneralAuthenticate` does not support a mechanism of first and non-first authentications, as multiple access conditions can be granted in one single authentication.

4.1.1 MAC Calculation

MACs which are used for EV2 Secure Messaging are calculated using the underlying block cipher according to CMAC standard described in Ref. [4].

The output of the in Ref. [4] described MAC calculation is a 16 byte MAC, which is further adopted for EV2 Secure Messaging usage. The 16 byte MAC is truncated to an 8 byte MAC, using only the even bytes in most significant order. The procedure is depicted in [Figure 2].

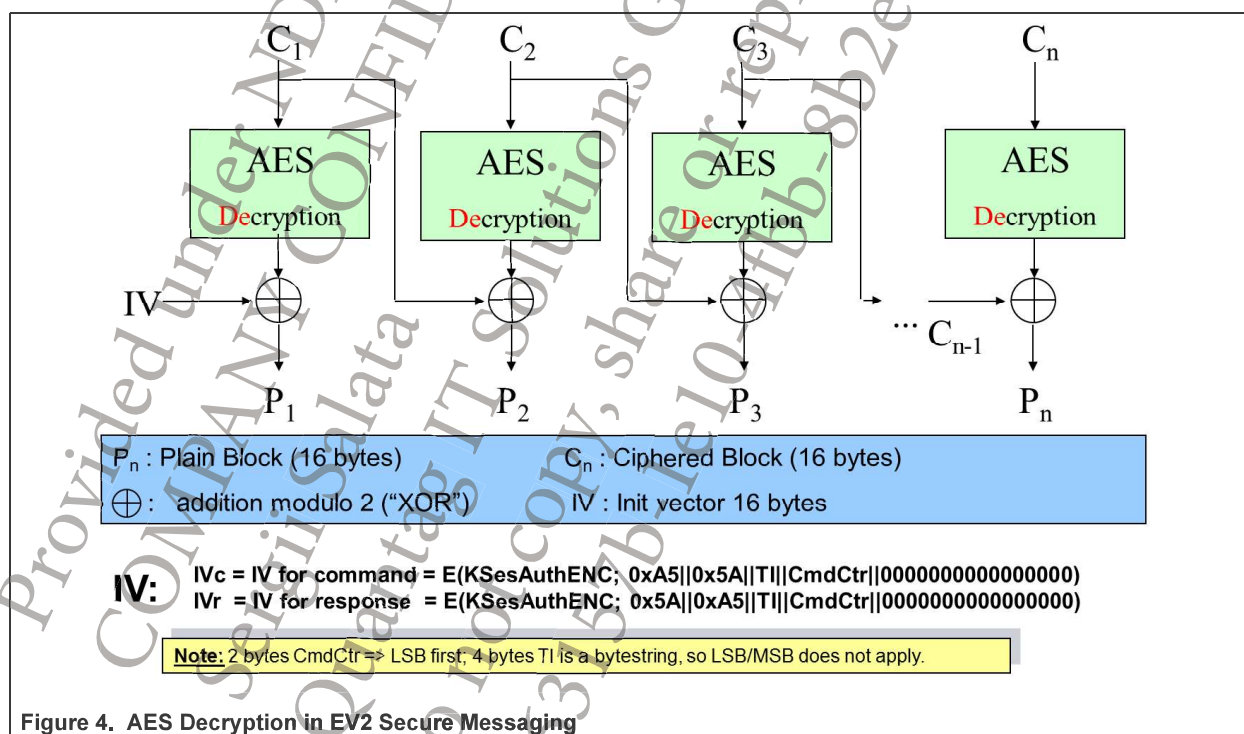
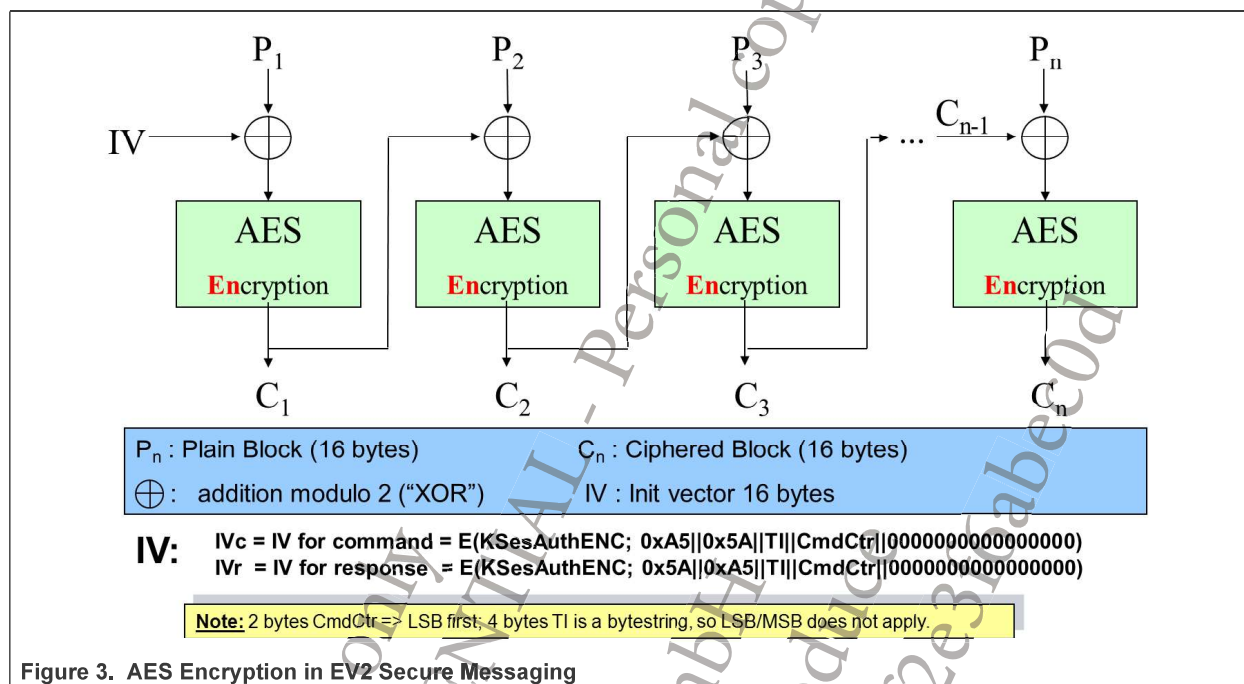


4.1.2 Encryption

Encryption and decryption are done using the underlying block cipher according to NIST Special Publication 800-38A Ref. [5]. Padding is done according to Padding Method 2 (0x80 followed by zero bytes) of ISO/IEC 9797-1. Note that if the original data is already a multiple of 16 bytes, another additional padding block (16 bytes) is added.

The IV (which is different for command and response) is always calculated as shown in [Figure 3]. The CmdCtr to be used in the IV always represents the current value, details to this see in Section 6.3.9.4 in Ref. [1].

For the encryption during authentication using Cmd.AuthenticateEV2First and Cmd.AuthenticateEV2NonFirst, the IV will always be a zero bytes IV with length 16.



4.1.3 Symmetric Authentication

As already mentioned shortly, two authentication commands `Cmd.AuthenticateEV2First` and `Cmd.AuthenticateEV2NonFirst` are available for EV2 Secure Messaging. The following [Table 8] shows the differences between the two authentication commands.

Table 8. Differences between `Cmd.AuthenticateEV2First` and `Cmd.AuthenticateEV2NonFirst`

	<code>Cmd.AuthenticateEV2First</code>	<code>Cmd.AuthenticateEV2NonFirst</code>
Goal	To start a new transaction	To start a new session within the ongoing transaction
Cmd	0x71	0x77
Capability bytes	PCD - PICC capability bytes are exchanged	No capability bytes are exchanged
Transaction Identifier	New transaction identifier is generated	No new transaction identifier is generated (old transaction identifier remains)
Counter	Counter is set to 0x0000	Counter remains as it is
Session keys	New session keys generated	

`Cmd.AuthenticateEV2First` can be used in any state (also when the PICC is in authenticated state) but `Cmd.AuthenticateEV2NonFirst` can be used only when the PICC is already in authenticated state. During a transaction, it is recommended to perform the subsequent authentications with `Cmd.AuthenticateEV2NonFirst`. This prevents the possibility of an interleaving attack (PICC works with two PCDs at the same time) as authentication with `Cmd.AuthenticateEV2NonFirst` does not generate a new transaction identifier.

Moreover the authentication using `Cmd.AuthenticateEV2NonFirst` is faster compared to `Cmd.AuthenticateEV2First`, as less bytes are exchanged.

Any of these two authentication methods can always be used for an authentication using the `KeyID.OriginalityKeys`. However, please note that an authentication using `KeyID.OriginalityKeys` will not lead the PICC to an authenticated state (necessary for other application or file level operations).

The mutual authentication of `Cmd.AuthenticateEV2First` as well as `Cmd.AuthenticateEV2NonFirst` is shown in [Figure 5].

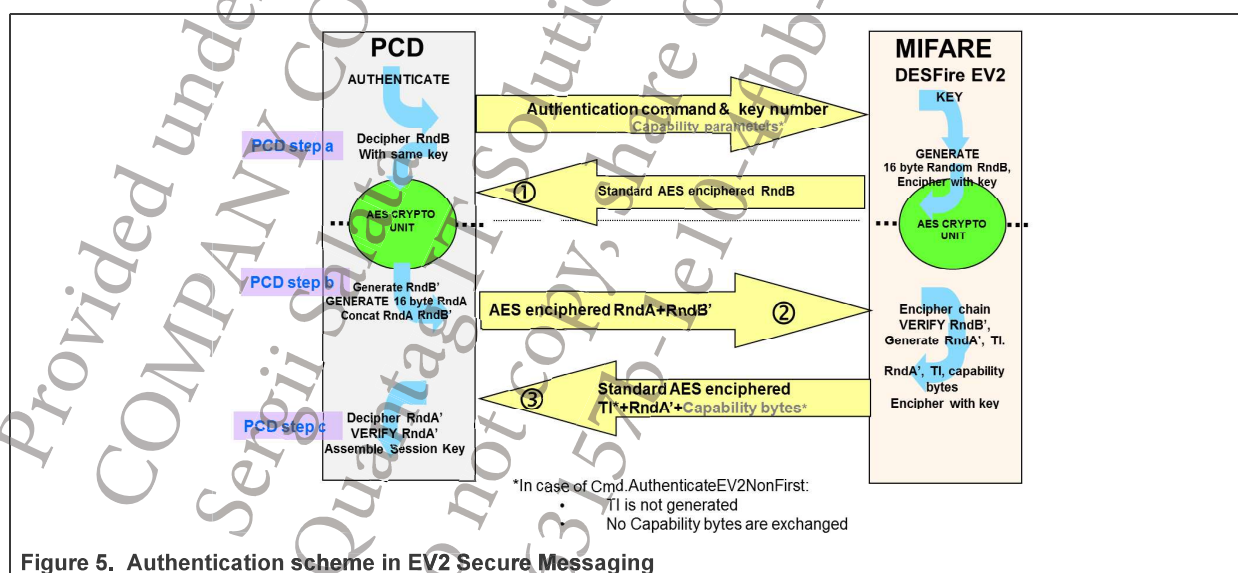


Figure 5. Authentication scheme in EV2 Secure Messaging

During the authentication, the PICC must accept messages from the PCD which are longer than the lengths derived from the specification as long as the parameter LenCap (length of the PCD capabilities) is correct. This needs to be done in order to support the upgradability features on future product versions. The PICC will also receive PCDcap2, but shall not interpret it at the moment.

After sending the response, the PCD will send the second part of the authentication command. This time the PICC will return 12 bytes of capabilities to the PCD – 6 bytes of PCD capabilities PCDcap2 (that were received on the first command) and 6 bytes of PICC capabilities PDcap2. The PCD needs to check the received capability bytes PCDcap2, and compare them to the originally sent ones.

The first four bytes of the PICC capabilities PDcap2 are reserved for the chip producer and set to 0x00. Bytes five and six can be configured using `Cmd.SetConfiguration`.

4.1.3.1 Example: Authentication using `Cmd.AuthenticateEV2First`

Key Nr = 0x00

Key = 00000000000000000000000000000000

Table 9. Example showing `Cmd.AuthenticateEV2First` authentication

Step	Command		Data Message
1	<code>Cmd.AuthenticateEV2First</code> (<code>Cmd KeyNo LenCap PDCap2</code>)	>	710000
2	R-Apdu ($E(K_x, \text{RndB})$)	<	AFC56F576D2444171CF64B196346A81662
3	Decrypted RndB IV = 00000000000000000000000000000000 000000	=	6CCC83FCBF582BA77D10B28025F224E6
4	PCD generates RndA	=	876D85B7FC717073AFBF564834F98F1E
5	PCD prepares RndB	=	CC83FCBF582BA77D10B28025F224E66C
6	RndA RndB	=	876D85B7FC717073AFBF564834F98F1ECC83FCBF582BA77D10B28025F224E66C
7	$E(K_x, \text{RndA} \text{RndB})$ IV = "00000000000000000000000000000000" 000000	=	60DCFF9A0BBE8DE18B7D79BB590CD4B42D531C24906D1B0D11BEB17E8850D298
8	<code>Cmd.AuthenticateEV2FirstPart2</code> C-APDU	>	AF60DCFF9A0BBE8DE18B7D79BB590CD4B42D531C24906D1B0D11BEB17E8850D298
9	R-APDU $E(K_x, \text{TI} \text{RndA}' \text{PDCap2} \text{PCSCap2})$	<	00EE93375DE2190A24F97D4AE363CAEC8DE2ED76DF4C3EE23C9D3499E3EC8D2259
10	$D(K_x, \text{TI} \text{RndA}' \text{PDCap2} \text{PCSCap2})$ IV = 00000000000000000000000000000000 000000	=	B04D6C116D85B7FC717073AFBF564834F98F1E87000000000000000000000000
11	PCD prepares RndA	=	876D85B7FC717073AFBF564834F98F1E
12	PCD compares sent and received RndA	=	876D85B7FC717073AFBF564834F98F1E ? 876D85B7FC717073AFBF564834F98F1E
13	TI	=	B04D6C11
14	PDCap2	=	000000000000
15	PCDCap2	=	000000000000

After the authentication, the two session keys KeyID.SesAuthENCKey and KeyID.SesAuthMACKey can be derived. The detailed steps for generating the session keys are explained in [Section 4.1.4](#). In this example, the session keys have the following values:

KeyID.SesAuthENCKey = 030A8C76BBC954513A52B2AAD161D15B

KeyID.SesAuthMACKey = 185D0E3CEA4C0C32DFAD84B3414A5054

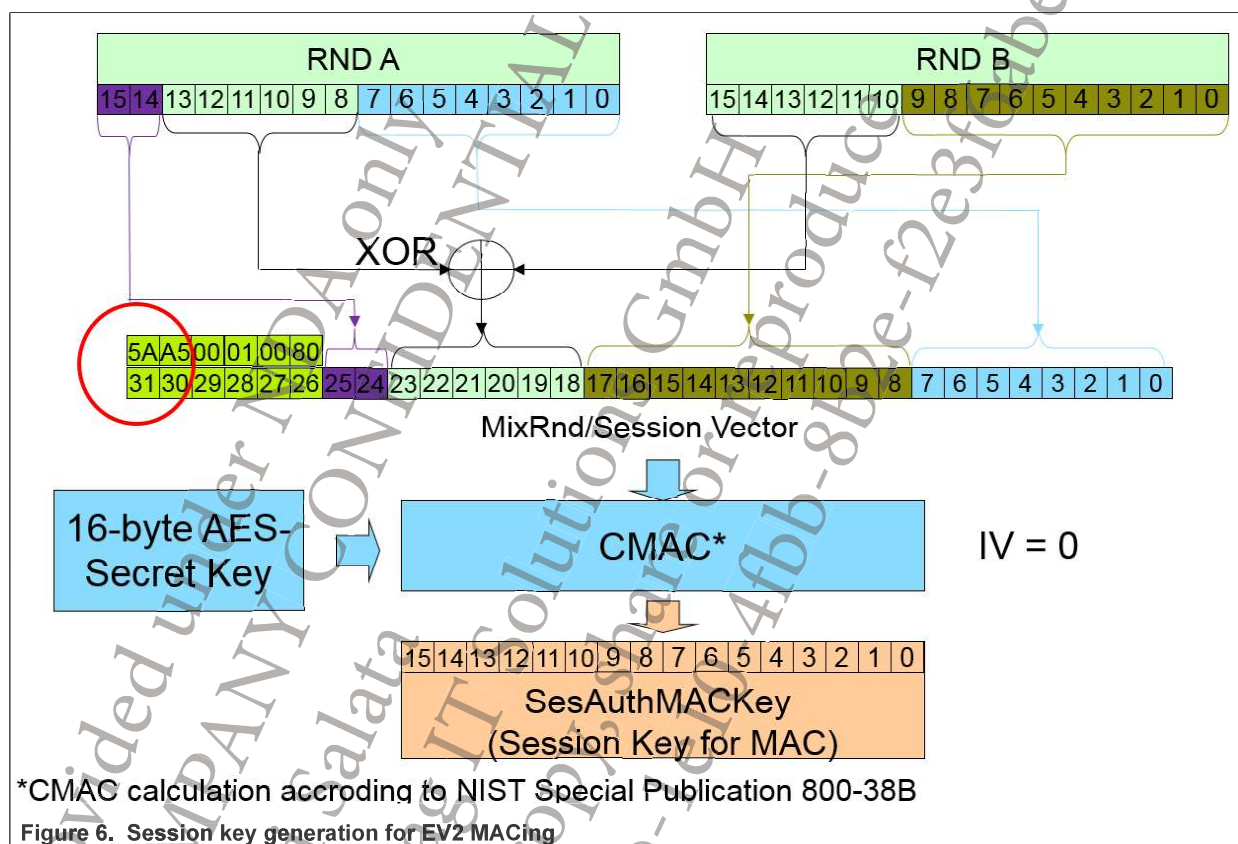
The used IV has been zero throughout the authentication process. The command counter was reset to zero.

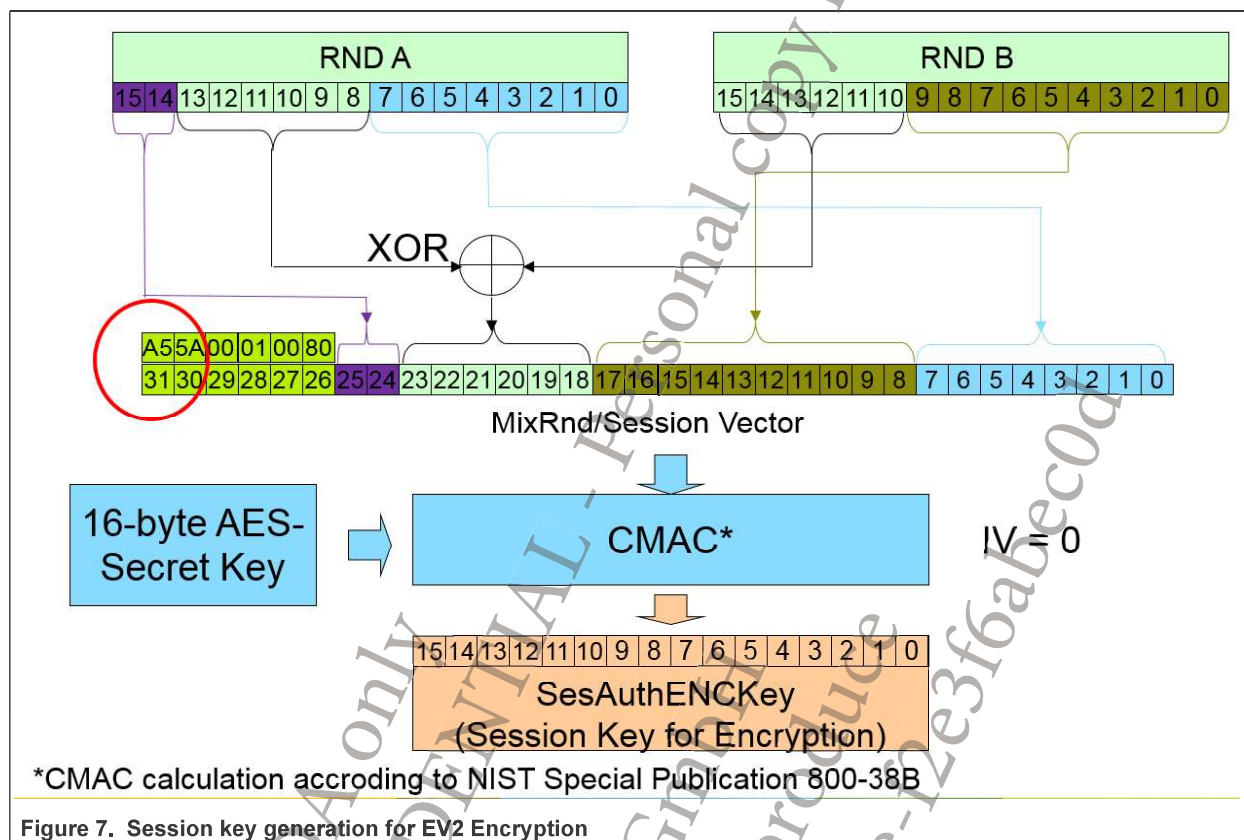
4.1.4 Session Keys Generation

After successful authentication with Cmd.AuthenticationEV2First or Cmd.AuthenticateEV2NonFirst, two session keys are generated both by the PICC and the PCD according to NIST SP 800-108 Ref. [6] as shown in [Figure 6](#) and [Figure 7](#). The two created session keys are:

KeyID.SesAuthMACKey or KSesAuthMAC for MACing of messages

KeyID.SesAuthENCKey or KSesAuthENC for encryption and decryption of messages





4.1.5 Asymmetric Authentication

MIFARE DUOX is the first product in the MIFARE family to support an asymmetric authentication based on ECC-256 key pairs. After an asymmetric authentication, the secure messaging will still be based on AES-128 keys, hence there is no difference in secure messaging after the authentication between asymmetric and symmetric authentications.

The asymmetric authentication is based on the well known Station-To-Station (STS) protocol for authenticated key agreement, which offers perfect forward security and optional mutual or reader-unilateral authentication via certificate based signatures.

The asymmetric authentication is performed with `Cmd.ISOGeneralAuthenticate`, and can be performed as either mutual or reader-unilateral authentication.

Similar as for a symmetric authentication based with `Cmd.AuthenticateEV2First` and `Cmd.AuthenticateEV2NonFirst`, an asymmetric authentication using `Cmd.ISOGeneralAuthenticate` can grant the reader specific access conditions for file access and other functionalities, but with the advantage that an asymmetric authentication can grant multiple access conditions at once. The access conditions obtained are specified with the corresponding `CARootKey` that is used during the authentication to certify the reader certificate. The configured access conditions of a `CARootKey` can be limited by the used certificate as well, giving the system operator various possibilities of restricting the access of single readers to card populations. More about this configurations options can be found in [Section 7](#).

The actual authentication procedure is again a 3-pass mutual authentication.

4.1.5.1 Example: Card personalization and mutual asymmetric authentication

Following example shows a mutual authentication using ECC keys. Additionally, the whole process of injecting the ECC Key and the CARootKey including the certificate is shown.

Following test keys and certificates were used in this transaction:

- [illegible]

Table 10. Asymmetric Authentication example

Step	Action	Direction	Command	Comment
1	Activate ISO14443-4 RATS	>	E080	
2	ATS	<	0678777810280	
3	Create Application	>	CA112233EF82	
4	Success	<	00	
5	GetApplicationIDs	>	6A	
6	Success	<	00112233	
7	SelectApplication	>	5A112233	
8	Success	<	00	

Table 10. Asymmetric Authentication example...continued

Step	Action	Direction	Command	Comment
9	AuthenticateEV2 First Part1	>	710000	Authenticate with the symmetric default key that is generated with the application
10	Response 1	<	AFC2608B2435091745EAB945422EB55E8C	
11	AuthenticateEV2 First Part2	>	AFB2BC27411F637A20087F9C0ED302163B86E9AD39A17A59A4BCBDC2D90C788ED3	
12	Response 2	<	00B47E279491117E9962749F7A67AC611323D3A57A4FC0E68828B1B3D516C820E3	
13	ManageKeyPair	>	4600010C80013000000000DB667013666476F61CD0659154EA2471EB586F0A5813387725B699D8AA073FC44C9F38F5402EB0056E96D7E6D76C7838621EA26DC0A26B36	ManageKeyPair: This command creates an asymmetric key pair in MIFARE DUOX. This example imports a already present private key into the asymmetric key location 0x00. The key can be used for asymmetric mutual authentication, as well as reader and card unilateral authentication.
14	Success	<	00DF624DF633FF561C	
15	ManageCARoot Key	>	48000C03003030000000009004CDB544E1858E5CD7E41D7C56D3648FE12387FD31A1B44F623AC04A9FA8A424496E0D842B48E2517E86B9C9B5C9B4B673EA874E7694DF76307838EF502C7DA129E72269385712E25F741F91FADE57EE32ECA6E9BE75E3	ManageCARoot Key loads the CA Root public key, that will be used for the reader certificate validation step during the asymmetric authentication.
16	Success	<	0099B34C563C1417CE	
17	CreateStdDataFile	>	CD0000EEEE0003002F874DD912889D97	Create a standard data file that will hold the certificate for the cards public key. This certificate needs to be signed with the CA root key which is available on the reader, i.e. the reader certifies the cards public key.
18	Success	<	00FFF513147D34AB12	

Table 10. Asymmetric Authentication example...continued

Step	Action	Direction	Command	Comment
19	WriteData (Certificate)	>	3D000000004A0100470100308201433081EAA0030201020 20ACCCCCCCCCCCCCCCCCCCCC300A06082A8648CE3D040302 301B31193017060355040A0C10	Write the card certificate into the previously generated StdDataFile. It is mandatory, that the first 3 bytes in the file give the length of the certificate (LSB First, here 470100)
20	AdditionalFrame	<	AF	
21	WriteData	>	AFCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC3022180F31393 730303130313030303030305A180F393939393132333132 33353935395A302F3114301206	
22	AdditionalFrame	<	AF	
23	WriteData	>	AF0355042D030B00CCCCCCCCCCCCCCCCCCCC31173015060 355040D0C0ECCCCCCCCCCCCCCCCCCCCCCCCCCCC30593013 06072A8648CE3D020106082A86	
24	AdditionalFrame	<	AF	
25	WriteData	>	AF48CE3D030107034200040889CA87F0E97C088C596F3F5 0EEC82D295549BB2B20956176F3B5B0B2EC5F8F55939695 8821F723A3C46684EC2B9BDB74	
26	AdditionalFrame	<	AF	
27	WriteData	>	AFD4F3B69D3F8F195EB0E9C8CA911836300A06082A8648C E3D0403010348003045022037DCF93C46E012AC2D14738D 04BB3D71BC7DDCE85DDA3A23AA	
28	AdditionalFrame	<	AF	
29	WriteData	>	AF2F3E31C25340A402210082389BEC11BCB0EC560D6C8AB 601183E5F800E7BD50E68185B3BE60665335EC3	
30	Success	<	00	

Table 10. Asymmetric Authentication example...continued

Step	Action	Direction	Command	Comment
31	ISOGeneral Authenticate Part1	>	008700004B80048000000007C43854104CA7D8D8830F3EF470BD2EB62A1A1A9289527F425820D86A9B991E7206816C2E6B6F39D8824CFD56DD2EEC68EED4E7B09F70870F1B8AA8EA25F61125E9B25947B00	ISOGeneral Authenticate performs a mutual asymmetric authentication.
32	Response 1	<	7C438541040E4022C213C687114D74FBC0F20EC3850A356734A5E709A1F8F37F6614500ECC62D6841EF96B3178BB1193805F0DA1D674E11F3E8C6814AD457527DF2329CF2E9000	
33	ISOGeneral Authenticate Part2	>	008700000001787C8201748682017013E1AC28897AA234A7D16390B351C670EEB1B4FAF914F0BC5BF146D89F41B316568E2FC3E86E1E8FD01D6D61B8F5DC88DE1FBACDB98BE697CC5E524C38C06A5E1836ED57AD25E4E3DDD0DB9FEB38C75A58FE1934A9DDCE6CCEACCE7F828CB33ABBBEA374C373A6EF2196F9F4A6CF7A5DA099FD213BC29D790339B0D93240E1FB16FD741E54257919E9FD1F741086A4A2C34D6001BB511DE7A4601D5B06E1FB8D3A889F7A4D150004F9D7C31A97C6F73E098493B6FCA4935818C5A763ADC5C3C9183EC4FDE84C8A352CA9AAF88B55C2C98877B16454B8133704A8F9657C69404FC106DB8FBAE50136634C5B6BD6D044CCBF7130598927DD67A550AA84692C4A9D037227CD081B3057B2756890B0B83C5ACFF5AE48078178AEFFDE774E931510852086DF2E8D937DF69174C61D030C189BD7AEBEFD6FCF54ED457ADA3C7A52A5E895E0612326D6197DDB977CE5FD2D2EBE33C3459B1F645846E39E424A15A05152FA3920401557378B243481E7FE502EC30000	
34	Response 2	<	7C8201A4828201A05980ADC660208DB284ABB4C68E181C9FF9A75E846505B73C956848C356E8BA9AD5E882940E2B53F39302DF3980C83875AEF18DEACD42003F2756F757FDE44015856A91EAF41C6EC673946C72FF8E1F3E662FF562338B1E7B6D327BC55E0605E5DE2D43B7C1141F839C07720C15D84EF9DF90262C3B4C9F047A9E80F337D25AD0F925D11A17E7FD991BF8E872862F45F2E8A4BDF4A6401179B834B63DCFA023820E554EBC91ED9A0E6A2F7C1AE2EFAD4B0E640507B77CA436811B9D1FECE949CC6C867C4EACFBE26A67997465F70C27768DB0942B174760910F7E01265E061FA826DB7A9CCD4CA26406FF91825E325D7BA45A2557C5907BC9F2E8181E1C593A22F5A1B02C1E7003C09CDE1B10A9B7E2A07D8DE2D51E2B03EB5F3D8E026E5F11D12A19C97DD2DB8985830463B2F5BE26297733BC64FF747152383F550905D25B68045F055E851716FAB45A8D5004B3480621381B864224527C516B816A56117FB779E726AF225D9BD278EDEA7D769ED8A2A07C6E4F2C21AE279EC3A633429ABD942BCADD68985CA3C98ED281CAEC51DFCCF72049D7326C21789084FE9F042E78499000	

4.1.5.2 Session Keys generation for ECC authentication

The session key generation for the ECC authentication is according to the Two-Step Key Derivation of [NIST SP800-56C](#).

In the first step, a KDK (Key Derivation Key) is generated out of a concatenation of the 8 least significant bytes from both shared ephemeral public keys **X-coordinates** during the first step of Cmd.ISOGeneralAuthenticate used as an AES key and the shared secret (ShS) from the ECDH key agreement as message.

$$KDK = MAC(E.Pub.A.x[7..0] || E.Pub.B.x[7..0], ShS)$$

As a second step, the KDK is expanded according [NIST SP 800-108](#) in Ctr-mode, where the pseudo-random function is again the AES-128 CMAC.

The input data is constructed using the following fields as defined by [NIST SP 800-108](#). Note that [NIST SP 800-108](#) allows defining a different order than proposed by the standard as long as it is unambiguously defined.

1. a 2-byte label, distinguishing the purpose of the key: 0x4BB4 for MACing and 0xB44B for encryption
2. a 2-byte counter, fixed to 0x0001 as only 128-bit keys are generated.
3. a 2-byte length, fixed to 0x0080 as only 128-bit keys are generated.
4. a 10-byte context, set to all 0x00

Therefore, the 16-byte session keys are constructed as follows:

$$K_{SesAuthENC} = MAC(KDK, 0xB4 || 0x4B || 0x00 || 0x01 || 0x00 || 0x80 || 0x00 || 0x00 || 0x00 || 0x00 || 0x00 || 0x00 || 0x00 || 0x00 || 0x00 || 0x00)$$

$$K_{SesAuthMAC} = MAC(KDK, 0xB4 || 0x4B || 0x00 || 0x01 || 0x00 || 0x80 || 0x00 || 0x00 || 0x00 || 0x00 || 0x00 || 0x00 || 0x00 || 0x00 || 0x00 || 0x00)$$

A numeric example with the data from [Table 10](#) is given below:

$$KDK = (B991E7206816C2E6F8F37F6614500BCC, DF7FD83547C1551B0A1E9A0421CFD6E1721D7FE4B51534652FEFA2222A775506)$$

$$KDK = 505BA954F6742378FC5331ADC3FB3D7B$$

Table 11. Numerical example of session key generation during asymmetric authentication

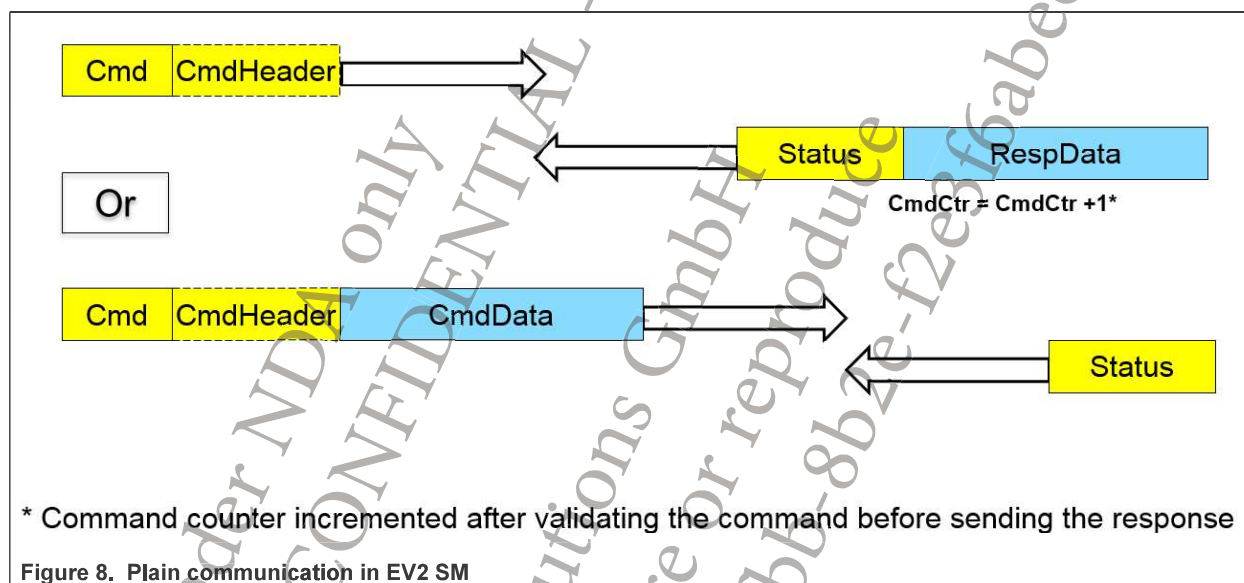
Value	Command	Comment
E.Pub.A (from step 31):	04	prefix, uncompressed byte representation
	CA7D8D8830F3EF470BD2EB62A1A1A9289527F425820D86A9B991E7206816C2E6	x-coordinate
	B6F39D8824CFD56DD2EEC68EED4E7B09F70870F1B8AA8EA25F61125E9B25947B	y-coordinate
E.Pub.B (from step 32):	04	prefix, uncompressed byte representation
	0E4022C213C687114D74FBC0F20EC3850A356734A5E709A1F8F37F6614500BCC	x-coordinate
	62D6841EF96B3178BB1193805F0DA1D674E11F3E8C6814AD457527DF2329CF2E	y-coordinate
ShS	DF7FD83547C1551B0A1E9A0421CFD6E1721D7FE4B51534652FEFA2222A775506	not shown in log
KDK	KDK = (B991E7206816C2E6F8F37F6614500BCC, DF7FD83547C1551B0A1E9A0421CFD6E1721D7FE4B51534652FEFA2222A775506)	
	505BA954F6742378FC5331ADC3FB3D7B	KDK

Table 11. Numerical example of session key generation during asymmetric authentication...continued

Value	Command	Comment
K _{SesAuthENC}	K _{SesAuthENC} = (KDK, B44 B00010080000000000000000000000000)	IV = 0000..00
	062CCA3C1B1A889164A21873B13D2C26	
K _{SesAuthMAC}	K _{SesAuthENC} = (KDK, 4 BB40001008000000000000000000000000)	IV = 0000..00
	070FA8F70C57AF7DE92AA386F1B0F04E	

4.1.6 CommMode.Plain

In this mode, data is sent between PCD and PICC in plain and is not secured. The command counter increment will be maintained, so that any subsequent command in `CommMode.MAC` or `CommMode.Full` will detect illegal insertion or deletion of commands.

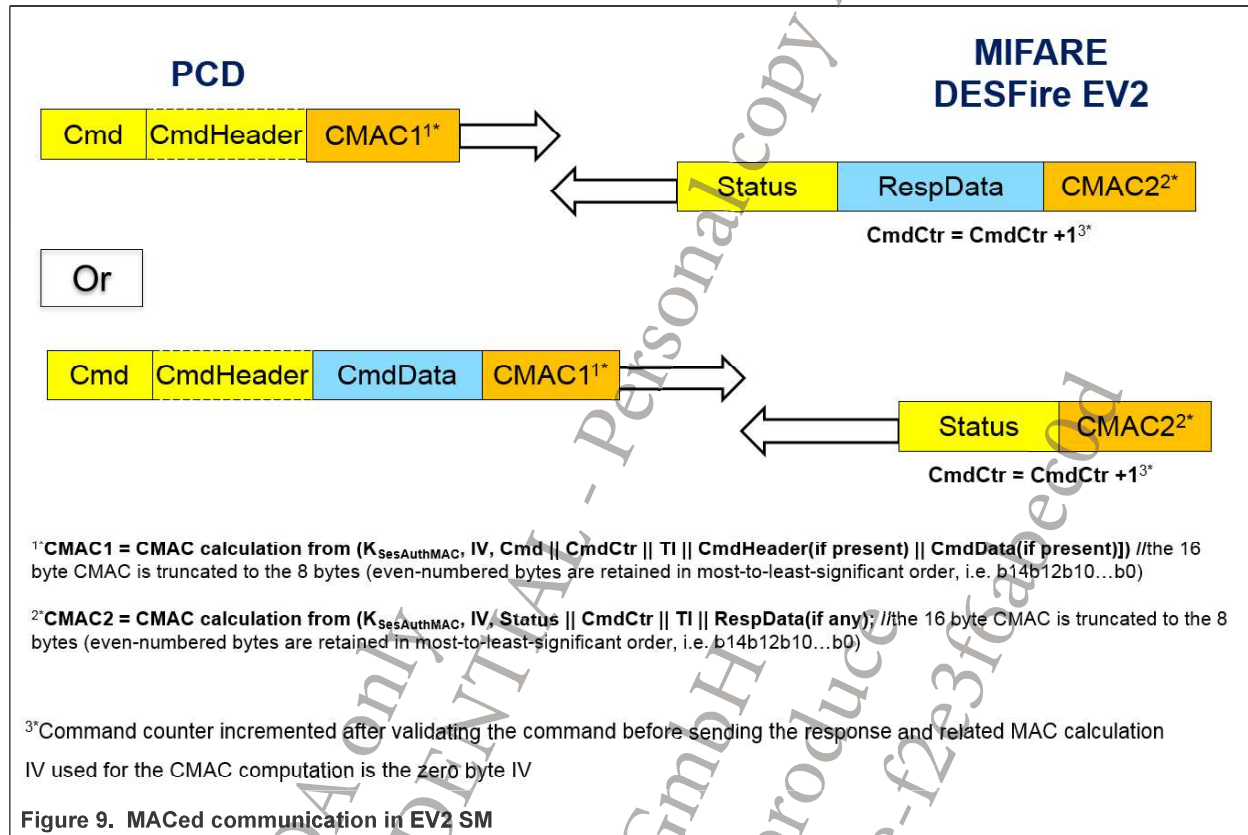


4.1.7 CommMode.MAC

In EV2 Secure Messaging MACed communication mode, an 8 byte MAC (CMAC) ensures the data integrity (calculation of the MAC is done according to [Section 4.1.1](#)). The session key which results after successful authentication, `KeyID.SesAuthMACKey`, is used for calculating the MAC. For the C-APDU, the MAC is calculated over the command, the command counter, the transaction identifier, the command header (if present) and the command data (if present). For the R-APDU, the MAC is calculated over the response code, the command counter, the transaction identifier and the response data (if present).

Note that the command counter is increased before sending the R-APDU and before doing the MAC calculation which is needed for the R-APDU.

A list of commands which require EV2 Secure Messaging in `CommMode.MAC` is available in Ref. [\[1\]](#).



4.1.7.1 Example: Cmd.WriteData using CommMode.MAC in EV2 Secure Messaging

KeyID.SesAuthMACKey = 185D0E3CEA4C0C32DFAD84B3414A5054

Current CmdCtr = 0100 (LSB first)

TI = B04D6C11

Table 12. Cmd.WriteData in EV2 Secure Messaging using CommMode.MAC

Step	Command	Data Message
1	Data to write	= 0402030405060708090A0B0C0D0E0F101112131415
2	File Nr	= 02
3	Offset	= 000000
4	Length of data	= 150000
5	IV	= 00000000000000000000000000000000
6	Input Data for calculating CMAC (Cmd CmdCtr TI FileNo Offset Length Data to write)	= 3D0100B04D6C11020000001500000102030405060708090A0 B0C0D0E0F101112131415
7	CMAC	= 16C090A3EB0889AA
8	Cmd.WriteData C-APDU	> 3D020000001500000102030405060708090A0B0C0D0E0 F10111213141516C090A3EB0889AA
9	Updated CmdCtr	= 0200

Table 12. Cmd.WriteData in EV2 Secure Messaging using CommMode.MAC...continued

Step	Command		Data Message
10	R-APDU	<	00377D8A81AF663484
11	Data for calculating CMAC on response (RC CmdCtr TI)	=	000200B04D6C11
12	CMAC to be received	=	377D8A81AF663484
13	PCD compares calculated CMAC and received CMAC	=	377D8A81AF663484 ? 377D8A81AF663484

Note: In Step 9, the CmdCtr gets updated after validating the command APDU, but before sending the response APDU. The updated CmdCtr is then used for the next calculation.

4.1.7.2 Example: Cmd.ReadData using CommMode.MAC in EV2 Secure Messaging

KeyID.SesAuthMACKey = 185D0E3CEA4C0C32DFAD84B3414A5054

Current CmdCtr = 0200 (LSB first)

TI = B04D6C11

Table 13. Cmd.ReadData in EV2 Secure Messaging using CommMode.MAC

Step	Command		Data Message
1	File Nr	=	02
2	Offset	=	000000
3	Length of Data to read	=	150000
4	IV	=	00000000000000000000000000000000
5	Input Data for calculating CMAC (Cmd CmdCtr TI FileNo Offset Length)	=	BD0200 B04D6C1102000000150000
6	CMAC to be sent		98871842F01DAB5F
7	Cmd.ReadData C-APDU	>	BD0200000015000098871842F01DAB5F
8	Updated CmdCtr	=	0300
9	R-APDU	<	0102030405060708090A0B0C0D0E0F1011121314153B14 CBFD05A8E327
10	Data for calculating CMAC on response (RC CmdCtr TI Response Data)	=	000300B04D6C110102030405060708090A0B0C0D0E0 F101112131415
11	IV	=	00000000000000000000000000000000
12	CMAC to be received	=	3B14CBFD05A8E327
13	PCD compares calculated CMAC and received CMAC	=	3B14CBFD05A8E327 ? 3B14CBFD05A8E327

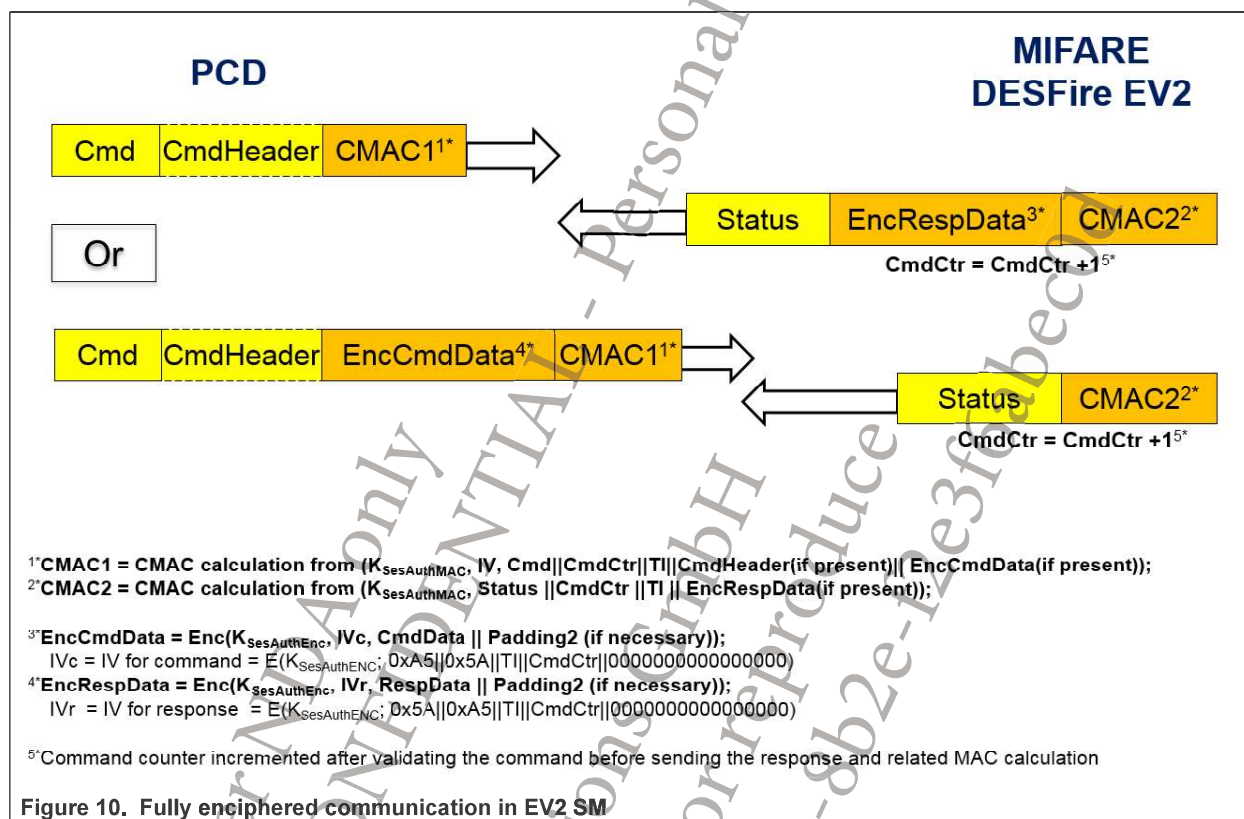
Note: In Step 8, the CmdCtr gets updated after validating the command APDU, but before sending the response APDU. The updated CmdCtr is then used for the next calculation.

4.1.8 CommMode.Full

In EV2 Secure Messaging Full communication mode, all encryption and decryption is done using the session key which results after successful authentication, KeyID.SesAuthENCKey. For the C-APDU, encryption is done

over the padded command data, for the R-APDU, encryption is done over the padded response data (detailed instructions for encryption can be found in [Section 4.1.2](#)).

Additionally to the encryption, a MAC is added to the encrypted APDU (command or response frame are handled like in CommMode.MAC). Using KeyID.SesAuthMACKey, a MAC is calculated and appended for transmission.



4.1.8.1 Example: Cmd.WriteData using CommMode.Full in EV2 Secure Messaging

KeyID.SesAuthENCKey = 030A8C76BBC954513A52B2AAD161D15B

KeyID.SesAuthMACKey = 185D0E3CEA4C0C32DFAD84B3414A5054

Current CmdCtr = 0500 (LSB first)

TI = B04D6C11

Table 14. Cmd.WriteData in EV2 Secure Messaging using CommMode.Full

Step	Command	Data Message
1	Data to write	= 0102030405060708090A0B0C0D0E0F101112131415
2	File Nr	= 03
3	Offset	= 000000
4	Length of Data	= 150000
5	Padded Data to make it a multiple of 16 bytes	= 0102030405060708090A0B0C0D0E0F101112131415800000000000000000000000

Table 14. Cmd.WriteData in EV2 Secure Messaging using CommMode.Full...continued

Step	Command		Data Message
6	$IV_{\text{command}} = E(K_{\text{SesAuthENC}}, 0xA5 \parallel 0x5A \parallel TI \parallel \text{CmdCtr} \parallel 0000000000000000)$	=	E82509A26010DBD17318F35E6E1E2830
7	Encrypted Data $E(K_{\text{SesAuthENC}}, \text{Padded Data})$	=	F7BB09DF5CF7C3BFF99F86FAE35E71EAA4FA4CC7D806DEBC4C49D42EAABC3A6F
8	Data for calculating CMAC (Cmd CmdCtr TI FileNo Offset Length Enc. Data)	=	3D0500 B04D6C1103000000150000 F7BB09DF5CF7C3BFF99F86FAE35E71EAA4FA4CC7D806DEBC4C49D42EAABC3A6F
9	IV for CMAC calculation	=	00000000000000000000000000000000
10	CMAC	=	34D21EC9D99A0B17
11	Cmd.WriteData C-APDU	>	3D03000000150000 F7BB09DF5CF7C3BFF99F86FAE35E71EAA4FA4CC7D806DEBC4C49D42EAABC3A6F34D21EC9D99A0B17
12 ^[1]	Updated CmdCtr	=	0600
13	R-APDU	<	00A9E45BF1857AF67B
14	Data for calculating CMAC on response (RC CmdCtr TI)	=	000600B04D6C11
15	CMAC to be received	=	A9E45BF1857AF67B
16	PCD compares calculated CMAC and received CMAC	=	A9E45BF1857AF67B ? A9E45BF1857AF67B

[1] In Step 12, the CmdCtr gets updated after validating the command APDU, but before sending the response APDU. The updated CmdCtr is then used for the next calculation.

4.1.8.2 Example: Cmd.ReadData using CommMode.Full in EV2 Secure Messaging

KeyID.SesAuthENCKey = 030A8C76BBC954513A52B2AAD161D15B

KeyID.SesAuthMACKey = 185D0E3CEA4C0C32DFAD84B3414A5054

Current CmdCtr = 0700 (LSB first)

TI = B04D6C11

Table 15. Cmd.ReadData in EV2 Secure Messaging using CommMode.Full

Step	Command		Data Message
1	File no	=	03
2	Offset	=	000000
3	Length of data to read	=	150000
4	IV	=	00000000000000000000000000000000
5	Data for calculating CMAC (Cmd CmdCtr TI FileNo Offset Length)	=	BD0700B04D6C1103000000150000
6	CMAC to be sent	=	BC9F9E75E6EB4014
7	C-APDU	>	BD03000000150000BC9F9E75E6EB4014
8 ^[1]	Updated CmdCtr	=	0800
9	R-APDU Encrypted Data 8 bytes CMAC	<	F4A604CD65F97290767B78B02A55A84648A13A2D5BC7E14C795B1F0AE4154711E359DA5BBEB0FE30

Table 15. Cmd.ReadData in EV2 Secure Messaging using CommMode.Full...continued

Step	Command		Data Message
10	Received CMAC	=	E359DA5BBEB0FE30
11	IV _{response} = E(K _{SesAuthENC} , 0x5A 0xA5 TI CmdCtr 0000000000000000) = E(5AA5B04D6C110800000000000000000000)	=	B26403916206E1FD4D5086C2244FB8AA
12	D(K _{SesAuthENC} , Response Data Padding)	=	0102030405060708090A0B0C0D0E0F1011121314158000000000000000000000
13	Data for calculating CMAC on response (RC CmdCtr TI Enc Data)	=	000800B04D6C11F4A604CD65F97290767B78B02A55A84648A13A2D5BC7E14C795B1F0AE4154711
14	CMAC to be received	=	E359DA5BBEB0FE30
15	PCD compares calculated CMAC and received CMAC	=	E359DA5BBEB0FE30 ? E359DA5BBEB0FE30

[1] In Step 8, the CmdCtr gets updated after validating the command APDU, but before sending the response APDU. The updated CmdCtr is then used for the next calculation.

4.1.9 EV2 Secure Messaging Summary

Table 16. EV2 Secure Messaging at a glance

Comm. setting	Access Right	Previous Auth.	Resulted communication
Plain	free	No	Plain (CmdCtr gets increased if an authentication was already established, otherwise CmdCtr starts counting)
Plain	free	Yes	
Plain	Key n	Must with key n	
MACed	Key n	Must with key n	CmdCtr gets increased PCD calculates CMAC and sends PICC response includes CMAC
Encrypted	Key n	Must with key n	CmdCtr gets increased PCD encrypts data (if data field is present) PCD always calculates CMAC and sends PICC response is encrypted data (if data field is present) and always includes CMAC

4.2 Secure Dynamic Messaging

Unlike the other three secure messaging modes mentioned earlier in this chapter, the Secure Dynamic Messaging (SDM) does not require any authentication to enable this secure messaging mode. In fact, the Secure Dynamic Messaging allows for confidential and integrity protected data exchange, without requiring a preceding authentication.

MIFARE DUOX supports SDM for reading from one or more Standard Data Files on the PICC. Secure Dynamic Messaging allows adding security to the data read, while still being able to access it with standard NDEF readers for NTAG Type 4 cards. The typical use case is an NDEF holding a URI and some meta-data, where SDM allows this meta-data to be communicated confidentiality and integrity protected towards a backend server.

The Secure Dynamic Messaging supported by MIFARE DUOX is compatible to the Secure Dynamic Messaging available in the NTAG 424 DNA product as well as the one from MIFARE DESFire EV3. Details to the NTAG 424 DNA and the built-in SDM feature can be found in [\[7\]](#).

4.2.1 SDM for Reading from a File

In this section, several options, parameters and characteristics of SDM will be explained.

SDM Read Counter

- 3 byte counter (SDMReadCtr)
- Can be enabled on a file basis independently (using the `ChangeFileSettings` command)
- When enabled for the file, the counter will be maintained, also if the counter is not mirrored in the PICCData
- The counter will be increased for every `ReadData` or `ISOReadBinary` command that's sent in non-authenticated state
- The counter is not increased when authenticated, i.e. during an ongoing secure messaging
- The counter is also not increased if several sequential read commands using the same command code are sent. However if another command is sent in between, the counter is increased again
- Is represented LSB first in binary encoding
- Is represented MSB first in ASCII encoding
- A `SDMReadCtrLimit` can be configured for the file
- If the `SDMReadCtr` reaches the `SDMReadCtrLimit` or 0xFFFFFFFF, an error will be returned to the read command
- The counter can be retrieved either mirrored as part of the `PICCData` or it can be read out using the `GetFileCounters` command

SDM Read Counter Limit

- 3 byte counter limit (`SDMReadCtrLimit`)
- Can be optionally set for a file
- Is a limit to restrict the number of reads that can be done to a file using Secure Dynamic Messaging
- Is represented LSB first in binary encoding
- Can be enabled using the `ChangeFileSettings` command
- Can be retrieved using the `GetFileSettings` command

PICC Data

- Holds metadata of the targeted PICC and file
- Consists of the VUID (UID of the PICC) and / or the SDMReadCtr
- PICCData can be plain or encrypted, this depends on the configuration of the access right: SDMMetaRead
 - If SDMMetaRead is set for free access (0xE), then the PICCData will be plain
 - If SDMMetaRead is set to an application key, then the PICCData will be encrypted
- PICCData can be mirrored within the file (and therefore mirrored to the read-out URL / NDEF message)
- PICCData mirroring is configured with the ChangeFileSettings command, defining the needed offsets
- In case of plain PICCData mirroring:
 - The mirror position of the VUID is defined by the VUIDOffset
 - Length of the VUID is 7 bytes = 14 ASCII chars (or alternatively for 10 byte long UIDs = 20 chars)
 - The mirror position of the SDMReadCtr is defined by the SDMReadCtrOffset
 - Length of the SDMReadCtr is 3 bytes = 6 ASCII chars
 - When both VUID and SDMReadCtr are mirrored, both are separated using the "x" character in NTAG2x. The same can also be achieved for MIFARE DUOX. Alternatively obviously other parameter separator characters can be freely chosen as well.
- In case of encrypted PICCData mirroring:
 - The mirror position of the encrypted PICCData is defined by the PICCDataOffset. No separate values need to be given for VUIDOffset and SDMReadCtrOffset, but both values are combined in one cryptogram, according to the SDM-specific PICCData encryption
- PICCData mirroring is only done in a non-authenticated state. If authenticated, no mirroring is done, i.e. the regular secure messaging is applied on the metadata

SDM Encrypted File Data

- SDM for reading supports mirroring the full or parts of the file data (file content, not metadata) encrypted
- The encrypted file data is called SDMEncFileData
- Not to be confused with PICCData or EncPICCData
- For encrypting the file data, the access right SDMFileRead needs to be configured to one available application key
 - If SDMFileRead is set to 0xF, mirroring of SDMEncFileData is not possible
- The SDMEncFileData is always mirrored within the file
- Mirroring position (SDMEncOffset) and length (SDMEncLength) are to be defined using the command ChangeFileSettings
- SDMEncLength needs to be a multiple of 32 bytes
- SDMEncFileData mirroring is only done in a non-authenticated state. If authenticated, no mirroring is done, i.e. the regular secure messaging is applied on file data

SDMMAC

- The SDMMAC is a MAC calculated over the response data
- SDMMACInputOffset defines from which position in the file the MAC calculation starts
- The SDMMAC is calculated using the SDMFileReadKey
 - If SDMFileReadKey is set to 0xF, the SDMMAC is not calculated and SDMMACOffset and SDMMACInputOffset are not present
- The SDMMAC is a 8 byte long truncated MAC (original length is a 16 byte long CMAC)
 - The even-numbered bytes are retained in most-to-least significant order
- If SDMMAC is mirrored in the file, the SDMMACInputOffset shall be smaller than or equal to SDMMACOffset
- As the SDMMAC is mirrored inside the file, it does not depend on the actual read command parameters (offset and length), but on the SDMMACInputOffset

4.2.2 SDM specific Encryption of PICCData

In case PICCData is mirrored encrypted, the so-called PICCDataTag specifies which metadata is mirrored.

The format encrypted PICCData is the following:

$$\text{EncPiccData} = \text{Enc}(K_{\text{SDMMetaRead}}, \text{PICCDataTag} \parallel \text{VCUID} \parallel \text{SDMReadCtr} \parallel \text{RandomPadding})$$

The random padding is generated for the response of the first read command. On subsequent read commands, targeting the same file, the same padding is reused.

Encryption of the PICCData is always done by the IC itself using the AES128 algorithm in CBC mode.

4.2.2.1 Example: Decryption of PICCData

For verification of the encrypted PICCData (EncPICCData), in the backend, on the terminal or inside an NFC mobile application, the following parameters need to be known.

Table 17. Preconditions for PICCData Decryption

<i>Prerequisites:</i>	SDMMetaReadKey that was used
<i>Offset name:</i>	PICCENCDDataOffset in URL
<i>Length [bytes]:</i>	PICCENCDDataLength
<i>Algorithm:</i>	$\text{PICCData} = \text{Dec}(K_{\text{SDMMetaRead}}, \text{PICCENCDData})$

An example of the decryption procedure of EncPICCData is shown in [Table 18](#).

Table 18. Decryption of PICCData

Step	Command	Data Message
1	Encrypted PICCData	= EF963FF7828658A599F3041510671E88
2	SDMMetaReadKey = Key 0x00	= 00000000000000000000000000000000
3	IV	= 00000000000000000000000000000000
4	$\text{Dec}(K_{\text{SDMMetaReadKey}}, \text{PICCENCDData})$	= C704DE5F1EACC0403D0000DA5CF60941
5	PICCDataTag	= C7
6	UID	= 04DE5F1EACC040
7	SDMReadCtr	= 3D0000
8	Random padding	= DA5CF60941
9	PICCDataTag [bit]	= 1100 0111
10	PICCDataTag - UID mirroring [bit7]	= 1 (UID mirroring enabled)
11	PICCDataTag - SDMReadCtr mirroring [bit6]	= 1 (SDMReadCtr mirroring enabled)
12	PICCDataTag - UID Length [bit3-0]	= 111b = 7d (7 byte UID)

4.2.3 SDM specific Encryption of File Data

When encrypting static file data, the encryption and decryption of the SDMEncFileData are calculated according to the AES128 algorithm in CBC mode.

The SDMEncFileData is defined as follows:

$SDMEncFileData = Enc(K_{SesSDMFileReadENC}, StaticFileData[SDMEncOffset \dots SDMEncOffset + SDMENCLength/2 - 1])$

The IV that's used for encryption is the following:

$IV = Enc(K_{SesSDMFileReadENC}, SDMReadCtr || 0x00000000000000000000000000000000)$

Encryption of the FileData is always done by the IC itself using the AES128 algorithm in CBC mode.

4.2.4 SDM specific SDMMAC Calculation

When applying SDM for reading, the SDMMAC is to be mirrored within the file.

The SDMMAC is defined as follows:

$SDMMAC = MACt(K_{SesSDMFileReadMAC}, DynamicFileData[SDMMACInputOffset \dots SDMMACOffset - 1])$

The SDMMAC is the truncated CMAC, calculated according NIST Special Publication 800-38B applying the EV2 secure messaging compliant truncation from 16 to 8 bytes.

The IV that's used for MACing is the zero byte IV.

4.2.5 SDM Session Key Generation

When using SDM for Reading, the session keys `SesSDMFileReadMACKey` and `SesSDMFileReadENCKey` need to be computed as follows:

$SV\ 1 = 0xC3 \parallel 0x3C \parallel 0x00 \parallel 0x01 \parallel 0x00 \parallel 0x80 \parallel VUID \parallel SDMReadCtr \parallel ZeroPadding$

$SV\ 2 = 0x3C \parallel 0xC3 \parallel 0x00 \parallel 0x01 \parallel 0x00 \parallel 0x80 \parallel VUID \parallel SDMReadCtr \parallel ZeroPadding$

$K_{SesSDMFileReadENC} = MAC(K_{SDMFileRead}; SV\ 1)$

$K_{SesSDMFileReadMAC} = MAC(K_{SDMFileRead}; SV\ 2)$

Session key generation is done according to NIST SP 800-108 in counter mode. The used algorithm is the CMAC algorithm as per NIST Special Publication 800-38B.

During Secure Dynamic Messaging for reading, the session keys are used in the following way:

- `KSesSDMFileReadMAC` is used for MACing of the file data
- `KSesSDMFileReadENC` is used for encryption of the file data

Note: These are not the same session keys as Secure Standard Messaging ones, which are generated during `AuthenticateFirst` or `AuthenticateNonFirst`.

Prerequisites: CMAC with AES-128 cipher core

Key used: `SDMFileReadKey`

Length [bytes]: 16

Algorithm:

1. $K_{SesSDMFileReadENC} = MAC(K_{SDMFileRead}; SV1)$
2. $K_{SesSDMFileReadMAC} = MAC(K_{SDMFileRead}; SV2)$

Output:

1. `KSesSDMFileReadENC`
2. `KSesSDMFileReadMAC`

Table 19. SDM Session Key Generation

Step	Command	Data
1	Is UID mirrored?	= If YES, it must be included in SV calculation
2	Is <code>SDMReadCtr</code> mirrored?	= If YES, it must be included in SV calculation
3	UID	= 04C767F2066180
4	<code>SDMReadCtr</code>	= 010000
5	<code>K_{SDMFileRead}</code>	= 5ACE7E50AB65D5D51FD5BF5A16B8205B
6	$SV1 = C33C\ 0001\ 0080\ [UID]\ [SDMReadCtr]\ [ZeroPadding]$ [1]	= C33C0001008004C767F2066180010000
7	$SV2 = 3CC3\ 0001\ 0080\ [UID]\ [SDMReadCtr]\ [ZeroPadding]$	= 3CC30001008004C767F2066180010000
8	IV	= 00000000000000000000000000000000
9	$K_{SesSDMFileReadENC} = MAC(K_{SDMFileRead}; SV1)$	= 66DA61797E23DECA5D8ECA13BBADF7A9
10	$K_{SesSDMFileReadMAC} = MAC(K_{SDMFileRead}; SV2)$	= 3A3E8110E05311F7A3FCF0D969BF2B48

[1] In case of encrypting file data - `PICCENCDATA`, mirroring of UID and `SDMReadCtr` is mandatory. Therefore, both are always included in SV1 calculation.

4.2.6 Enabling Secure Dynamic Messaging

SDM can be activated for one of multiple Standard Data Files within an application.

The enablement of SDM happens during the file creation using the CreateStdDataFile command. At time of file creation, the SDM needs to be enabled by setting FileOption bit6 to 1. An example for enabling SDM can be seen in [Section 4.2.6.1](#).

After SDM was already enabled during file creation, the SDM specific settings and configurations need to be set using the ChangeFileSettings command. Using this command also FileOption bit6 needs to be set to 1, indicating that SDM will be configured. How the command can be used for SDM configurations is shown in the next sections for different examples.

4.2.6.1 Example: Enabling SDM during Standard Data File creation

Cmd = CD

File ID = 05

File Option = 40 (additional access rights disabled, CommMode = plain, Secure Dynamic Messaging and Mirroring enabled)

AccessRights = 3012 (Read: 0x01, Write: 0x02, ReadWrite: 0x03, Change: 0x00)

File Size = 800000

Table 20. Create Standard Data File with SDM enabled

Step	Command		Data Message
1	Cmd.SelectApplication C-APDU (AID = 0x333333)	>	5A333333
2	R-APDU	<	00
7	Cmd.CreateStandardDataFile C-APDU	>	CD05403012800000
8	R-APDU	<	00

After successful creation of the file, the SDM is enabled for this specific file.

4.2.7 NDEF Formatting of a MIFARE DUOX application

The NFC Forum is a standardization consortium that was formed to advance the use of Near Field Communication technology by developing specifications, ensuring interoperability among devices and services, and educating the market about NFC technology.

The NFC Forum has defined a data format called NDEF to store different kind of application data. NDEF structured data may be stored inside contactless tags. In the NFC Forum, the "NFC Forum Type 4 Tag Operation" has been developed to describe how the reader device (called NFC Forum device) can store NDEF data inside an NFC Forum Type 4 Tag platform. The NXP product MIFARE DUOX is compatible with the "NFC Forum Type 4 Tag Operation" technical specification and the Type 4 Tag platform.

In the following sections it will be explained, how a MIFARE DUOX IC can be very simply formatted to work as "NFC Forum Type 4 Tag". One very basic configuration and creation of the NDEF files is shown, to give an idea of the formatting procedure. This document does not go into details of NFC Tag Type 4 specified states and proper protection / setup of the NDEF tag, but just outlines a minimum personalization sequence.

4.2.7.1 NDEF Formatting: Creating the NDEF application

For adding the NFC Forum Type 4 Tag functionality to a MIFARE DUOX IC, the files which are required for creating the NDEF structure need to be put either into a standalone / specific application, like in this example, or need to be added to an existing application additionally. This is up to the choice of the application provider.

In this example, a new application is being created with settings that are needed for realizing the NDEF capabilities (ISO File ID, ISO DFName, ..).

Table 21. Creating the NDEF application

Step	Command		Data Message
1	AID	=	0x000001
2	KeySettings1	=	0x09
3	KeySettings2	=	0xA6
4	ISOFileID	=	0x01E1
5	ISODFName	=	0xD2760000850101
6	C-APDU Cmd.CreateApplication	>	CA00000109A601E1D2760000850101
7	R-APDU	<	00 (SUCCESS)

After the application was successfully created, it needs to be selected for being able to add files into the application and to work on the NDEF Tag Type 4 structure.

Table 22. Selecting the NDEF application

Step	Command		Data Message
1	AID	=	0x000001
2	C-APDU Cmd.SelectApplication	>	5A000001
3	R-APDU	<	00 (SUCCESS)

Once the application is selected, an authentication needs to be executed with the Application Master key, for being authorized to create and add files to the NDEF application.

As the keys have not been changed yet, the key values are all set to the default zero byte keys. The key type of the application is AES. As authentication command either AuthenticateAES or AuthenticateEV2First can be chosen.

In this example, the AuthenticateEV2First command is used to authenticate with the Application Master key, KeyNr 0x00.

Table 23. Authentication with AuthenticateEV2First using the Application Master key of the NDEF application

Step	Command		Data Message
1	KeyNr	=	0x00
2	Key Value	=	0x00000000000000000000000000000000
3	C-APDU Cmd.AuthenticateEV2First	>	0x710000
4	R-APDU	<	AF58726A55EDFCCDA69FBA212DA81B49D9
5	C-APDU Cmd.AuthenticateEV2First Part2	>	AFCAC40E6DBE8E4C97453AEE6A29A2D9502EBA6CE4105 BA87E8091ED966A0A10F0
6	R-APDU	<	00A70331D1256E6A7CEFCB670571AAEDD73883D6F569C9 DC0586CACDBDA0F7D660
7	MAC Verification	=	SUCCESS, not shown here
8	Response Code	=	00 (SUCCESS)
9	Session Encryption Key (KSesAuthENC)	=	0x436F05BB8D3669466906C790C84FE6AB
10	Session MAC Key (KSesAuthMAC)	=	0xDBA0236E288BDA91C13B15EA52474F33
11	IV	=	0x00000000000000000000000000000000
12	TI	=	0x44815F2D
13	CmdCtr	=	0x0000

After the successful authentication, two session keys are derived both on card and on reader side, for data encryption for the following session, and establishing a secure channel.

Intermediate steps of the authentication (encryption, decryption, MAC calculations) are not shown, but can be found explained in detail in [Section 4.1.3.1](#).

4.2.7.2 NDEF Formatting: Creating the CC File

When creating an NFC Forum Type 4 Tag, the minimum requirement for realizing the required structure is to have a CC File (capability container) and an NDEF file available inside the NDEF application.

Steps for creating the CC File are displayed in [Table 24](#).

Precondition for the file creation is a valid authentication with the AppMasterkey (if required by the AppKeySettings). The authentication that was established in this case was initiated by the AuthenticateEV2First command as detailed in the previous section.

Table 24. Creating the CC File

Step	Command	Data Message
1	Session Encryption Key (KSesAuthENC)	= 0x436F05BB8D3669466906C790C84FE6AB
2	Session MAC Key (KSesAuthMAC)	= 0xDBA0236E288BDA91C13B15EA52474F33
3	IV	= 0x00000000000000000000000000000000
4	TI	= 0x44815F2D
5	CmdCtr	= 0x0000
6	File Nr	= 0x01
7	ISOFileID	= 0x03E1
8	FileOption	= 0x00
9	Access Rights	0xEEEE (just for the Initialized State, for easier personalization needs to be configured later on) 0x23E2 (R = Free, W = 0x02, RW = 0x02, Change = 0x03)
10	FileSize	= 0x3C0000
11	Cmd MAC Calculation MAC_Input = Cmd CmdCtr TI Cmd Header	= CD000044815F2D0103E100EEEE3C0000
12	MAC = CMAC(KSesAuthMAC, MAC_Input)	= B1E079248D5B2962E974A9D2D2C53C00
13	Truncated MAC	= E0245B6274D2C500
14	C-APDU Cmd.CreateStandardDataFile	> CD0103E100EEEE3C0000061A17F6D04D31DC
15	R-APDU	< 0093ABCCEE7136819A
16	Response MAC	= 93ABCCEE7136819A
17	Response MAC Calculation MAC_Input = Resp Data CmdCtr + 1 TI	= 00010044815F2D
18	MAC = CMAC(KSesAuthMAC, MAC_Input)	= AB93AAAB24CC12EE0D71EA3604815C9A
19	Truncated MAC	= 93ABCCEE7136819A
20	Calculated MAC != Response MAC	= 93ABCCEE7136819A == 93ABCCEE7136819A (SUCCESS)
21	Response Code	= 00 (SUCCESS)
22	CmdCtr	= 0x0100

4.2.7.3 NDEF Formatting: Creating the NDEF File

The next part of the NDEF formatting procedure is the creation of the NDEF File. The NDEF File will later on hold the NDEF message that shall invoke any automated action (URL, vCard, etc).

Steps for NDEF File creation are displayed in [Table 25](#).

Precondition for the file creation is a valid authentication with the AppMasterkey (if required by the AppKeySettings). The authentication that was established in this case was initiated by the AuthenticateEV2First command as detailed in the previous section.

Table 25. Creating the NDEF File

Step	Command		Data Message
1	Session Encryption Key (KSesAuthENC)	=	0x436F05BB8D3669466906C790C84FE6AB
2	Session MAC Key (KSesAuthMAC)	=	0xDBA0236E288BDA91C13B15EA52474F33
3	IV	=	0x00000000000000000000000000000000
4	TI	=	0x44815F2D
5	CmdCtr	=	0x0100
6	File Nr	=	0x02
7	ISOFileID	=	0x04E1
8	FileOption	=	0x40 (SDM enabled)
9	Access Rights	=	0xEEEE (just for the Initialized State, for easier personalization needs to be configured later on) 0x23E2 (R = Free, W = 0x02, RW = 0x02, Change = 0x03)
10	FileSize	=	0xC80000
11	Cmd MAC Calculation MAC_Input = Cmd CmdCtr TI Cmd Header	=	CD010044815F2D0204E140EEEEC80000
12	MAC = CMAC(KSesAuthMAC, MAC_Input)	=	348ED21C8F0C0ABD16A87AA27C17BB0C
13	Truncated MAC	=	8E1C0CBDA8A2170C
14	C-APDU Cmd.CreateStandardDataFile	>	CD0204E140EEEEC800008E1C0CBDA8A2170C
15	R-APDU	<	001EB1DCE0340E14A6
16	Response MAC	=	1EB1DCE0340E14A6
17	Response MAC Calculation MAC_Input = Resp Data CmdCtr +1 TI	=	00020044815F2D
18	MAC = CMAC(KSesAuthMAC, MAC_Input)	=	DF1EFDB19DDC85E0EB34450EB81474A6
19	Truncated MAC	=	1EB1DCE0340E14A6
20	Calculated MAC == Response MAC	=	1EB1DCE0340E14A6 == 1EB1DCE0340E14A6 (SUCCESS)
21	Response Code	=	00 (SUCCESS)
22	CmdCtr	=	0x0200

4.2.7.4 NDEF Formatting: Writing the CC File content

The CC file is the so-called capability container of the NDEF application. Inside the data part it's linking to the NDEF file and so making the Tag Type 4 formatting possible.

One example is outlined in [Table 26](#).

Table 26. Writing the CC File content

Step	Command		Data Message
1	File Nr	=	0x01
2	Offset	=	0x000000
3	Data	=	0x000F20003B00340406E10400C80000
4	Data Length	=	0x0F0000
5	C-APDU Cmd.WriteData	>	3D010000000F0000000F20003B00340406E10400C80000
6	R-APDU	<	00 (SUCCESS)

4.2.7.5 NDEF Formatting: Changing the file settings of the CC File

In the following step, the file settings of the CC file will be adapted, to set the access rights to specific key values.

The command that is needed, is the `ChangeFileSettings` command. We can exchange this in plain, as the Change access right of the file is currently set to 0xE, which means that no authentication and no secure messaging is needed for configuration changes.

Table 27. Changing the file settings of the CC File

Step	Command		Data Message
1	File Nr	=	0x01
2	ISOFileID	=	0x03E1
3	New Access Rights	=	0x23E2 (R = Free, W = 0x02, RW = 0x02, Change = 0x03)
4	Comm Mode	=	Plain
6	C-APDU Cmd.ChangeFileSettings	>	5F010023E2
7	R-APDU	<	00 (SUCCESS)

4.2.7.6 NDEF Formatting: Writing the NDEF File content and configuring it

The last step of the NDEF formatting procedure is the writing of the actual NDEF file content, meaning the NDEF message (URL, vCard, etc.).

In the following chapters, several examples of NDEF URLs will be given, in combination how they can be written to make use of Secure Dynamic Messaging.

Following examples will be elaborated in detail:

- Creating an NDEF URL plus mirroring plain PICCData (VUID, ReadCtr) attached to the URL, as explained in [Section 4.2.8.1](#)
- Creating an NDEF URL plus mirroring encrypted PICCData (VUID, ReadCtr) attached to the URL, as explained in [Section 4.2.8.2](#)
- Creating an NDEF URL plus mirroring encrypted file content attached to the URL, as explained in [Section 4.2.8.3](#)

4.2.8 Writing the NDEF content to the NDEF application

4.2.8.1 Example: SDM Mirroring with plain PICCData

The following example shows how to write a weblink (URL) into the NDEF file.

The URL that is used here is <https://www.nxp.com/index.html>

Additionally the SDM shall be enabled and effectively used for this file.

In this example, the following PICCData parameters will be mirrored to the URL. The PICCData consists of UID and SDMMReadCounter. UID, SDMMReadCounter and SDMMAC are always co-existing, meaning that by enabling / disabling PICCData mirroring, all are mirrored or not. Their mirror offsets within NDEF can be individually chosen.

- VCUID
- SDM Read Counter
- SDMMAC

All parameters will be mirrored in plain text (not encrypted) in this example. Applying the mapping, we want to have a URL structure like below:

<https://www.nxp.com/index.html?u=0000000000000000&c=000000&m=0000000000000000>

With this URL we are linking to the target website, and additionally attaching mirroring placeholders for the UID (&u=), for the SDM Read Counter (&c=) and additionally for the SDM MAC (&m=). The placeholders are initially filled with zeros, but will be updated on each tap of the IC automatically. Once reading out the file content, the parameters will reflect the updated values.

Now that the mirroring parameters and URL structure are defined, two steps need to be done for configuration completion:

- Writing of the NDEF URL to the NDEF file
- Adapting the file settings including the SDM mirroring configuration for the NDEF file

To enable the mentioned settings, the `ChangeFileSettings` command needs to be used. For this command, the `SDMOptions` need to be set to `0x40`.

4.2.8.1.1 Writing the NDEF File content

The following example shows how to write a weblink (URL) into the NDEF file.

The URL that is used here is "https://www.nxp.com/index.html?u=0000000000000000&c=000000&m=0000000000000000", pointing to the NXP website.

Table 28. Writing the NDEF File content

Step	Command	Data Message
1	File Nr	= 0x02
2	IV	= 0x0772C3A3A34ABC4352C60185F2A2BDE2
3	Data (NDEF header + URL in hex)	= 0x005AD10156550068747470733A2F2F777777E6E78702E6 36F6D2F696E6465782E68746D6C3F753D3030303030303030 30303030303026633D303030303030303030303030303030 66D3D30
4	Length	= 0x5C0000

Table 28. Writing the NDEF File content...continued

Step	Command		Data Message
5	Offset	=	0x000000
6	C-APDU Cmd.WriteData	>	0x3D020000005C0000005AD10156550068747470733A2F2F77 77772E6E78702E636F6D2F696E6465782E68746D6C3F753D 303030303030303030303030
7	R-APDU	<	0xAF (Additional Frame)
8	C-APDU	>	0xAF303026633D3030303030303030303030303030266D 3D30303030303030303030303030303030
9	R-APDU	<	00 (SUCCESS)

4.2.8.1.2 Changing the file settings of the NDEF File

In the following steps, the file settings of the NDEF file will be adapted, to configure the mirroring options.

As already outlined in [Section 4.2.8.1](#), the following URL with the defined mirroring placeholders shall be realized:

<https://www.nxp.com/index.html?u=0000000000000000&c=000000&m=0000000000000000>

File Data Content:

0x0050D1014C550068747470733A2F2F7777772E6E78702E636F6D2F696E6465782E6
8746D6C3F753D30303030303030303030303026633D303030303030266D3D30303
0303030303030303030303030

NDEF File starting bytes: 0x0050D1014C5500

URL bytes: 0x68747470733A2F2F7777772E6E78702E636F6D2F696E6465782E68746D
6C3F753D3030303030303030303030303026633D303030303030266D3D303030303
03030303030303030

Table 29. Changing the file settings of the NDEF File

Step	Command		Data Message
1	File Nr	=	0x02
2	FileOption	=	0x40 (SDM enabled, CommMode Plain)
3	New Access Rights	=	0x23E2 (R = Free, W = 0x02, RW = 0x02, Change = 0x03)
4	NrAdditional Access Rights	=	n.a.
5	Additional Access Rights	=	n.a.
6	SDMOptions	=	0xC1 (VUID mirroring enabled, SDMReadCtr enabled, ASCII encoding mode)
7	SDMAccessRights	=	0xF3E3 SDMMetaReadKey = 0xE (Plain PICCData mirroring) SDMFileReadKey = 0x3 SDMContrKey = 0x3
8	VUID Offset	=	0x280000

Table 29. Changing the file settings of the NDEF File ...continued

Step	Command		Data Message
9	SDMReadCtr Offset	=	0x390000
10	PICCDData Offset	=	n.a.
11	SDMMACInput Offset	=	0x420000 (means MAC calculation of the empty string)
12	SDMENC Offset	=	n.a.
13	SDMENC Length	=	n.a.
14	SDMMAC Offset	=	0x420000
15	SDMReadCtr Limit	=	n.a.
16	TMC Limit	=	n.a.
17	C-APDU Cmd.ChangeFileSettings	>	5F024023E2C1F3E3280000390000420000420000
18	R-APDU	<	00 (SUCCESS)

4.2.8.1.3 NDEF Validation: CMAC Calculation

SDMFileReadKey Nr = 0x3

SDMFileReadKey value = 0x11112222333344445555666677778888

Data content that was read from the file using SDM =

0x0050D1014C550068747470733A2F2F7777772E6E78702E636F6D2F696E6465782E6
8746D6C3F753D303043393838363331343031303626633D303030303038266D3D39344
33041383345454342353737443030303030303030303000000000000000000000000
00
00
00

ASCII Text of the NDEF URL =

https://www.nxp.com/index.html?u=00C98863140106&c=000008&m=94C0A83EECB577D0

In order to be able to verify the SDMMAC that was mirrored to the URL, as first step the SDM session keys need to be calculated, as outlined in [Section 4.2.5](#). After the session keys have been calculated, the SesSDMFileReadMAC can be used for validating the SDMMAC.

Table 30. SDM Session Key Generation

Step	Command		Data
1	Is UID mirrored?	=	YES (it must be included in SV calculation)
2	Is SDMReadCtr mirrored?	=	YES (it must be included in SV calculation)
3	UID	=	00C98863140106
4	SDMReadCtr	=	080000
5	KSDMFileRead	=	11112222333344445555666677778888

Table 30. SDM Session Key Generation...continued

Step	Command		Data
6	SV1 = C33C 0001 0080 [UID] [SDMReadCtr] [ZeroPadding] ^[1]	=	C33C0001008000C98863140106080000
7	SV2 = 3CC3 0001 0080 [UID] [SDMReadCtr] [ZeroPadding]	=	3CC30001008000C98863140106080000
8	K _{SesSDMFileReadENC} = MAC(K _{SDMFileRead} , SV1)	=	0FE264FC0B406A5E62A2908BE1C899ED
9	K _{SesSDMFileReadMAC} = MAC(K _{SDMFileRead} , SV2)	=	025C33693BE8128815EBF30DC65A5D3A

[1] In case of encrypting file data - PICCENCDData, mirroring of UID and SDMReadCtr is mandatory. Therefore, both are always included in SV1 calculation.

Table 31. SDMMAC Validation

Step	Command		Data
1	Mirrored CMAC in URL	=	0x94C0A83EECB577D0
2	K _{SesSDMFileReadMAC}	=	0x025C33693BE8128815EBF30DC65A5D3A
3	SDMMACInput Offset	=	0x420000
4	SDMMAC Offset	=	0x420000
5	SDMMACInput Offset == SDMMAC Offset	=	YES, therefore SDMMAC calculation over the empty string
6	IV	=	0x00000000000000000000000000000000
7	SDMMAC Input	=	"" (empty string)
8	Calculated SDMMAC	=	0x94C0A83EECB577D0
9	Calculated SDMMAC == Mirrored SDMMAC	=	True (SUCCESS)

4.2.8.2 Example: SDM Mirroring with encrypted PICCData

The following example shows how to write a weblink (URL) into the NDEF file.

The URL that is used here is <https://www.nxp.com/index.html>.

Additionally the SDM shall be enabled and effectively used for this file.

In this example, the following PICCData parameters will be mirrored encrypted to the URL. The PICCData consists of UID and SDMReadCounter. UID, SDMReadCounter and SDMMAC are always co-existing, meaning that by enabling / disabling PICCData mirroring, all are mirrored or not. Their mirror offsets within NDEF can be individually chosen.

- Enc(KSDMMetaRead, VCUID || SDM Read Counter || Random Padding)
- SDMMAC

Note: With encryption of the PICCData, UID and SDMReadCounter are encrypted. Therefore the verification side does not have immediate info on UID, which is usually used as input for key derivation functions. In this case, KSDMMetaRead key shall not be UID diversified and high attention needs to be put on secure storage on system level of this key.

Applying the mapping, we want to have a URL structure like below:

<https://www.nxp.com/index.html?enc=00000000000000000000000000000000&mac=0000000000000000>

With this URL we are linking to the target website, and additionally attaching mirroring placeholders for the encrypted PICCData (&enc=) and for the SDM MAC (&mac=). The placeholders are initially filled with zeros, but will be updated on each tap of the IC automatically / dynamically. Once reading out the file content, the parameters will reflect the updated values.

Now that the mirroring parameters and URL structure are defined, two steps need to be done for configuration completion:

- Writing of the NDEF URL to the NDEF file
- Adapting the file settings including the SDM mirroring configuration for the NDEF file

To enable the mentioned settings, the `ChangeFileSettings` command needs to be used.

4.2.8.2.1 Writing the NDEF File content

The following example shows how to write a weblink (URL) into the NDEF file.

The URL that is used here is <https://www.nxp.com/index.html?enc=00000000000000000000000000000000&mac=0000000000000000>, pointing to the NXP website and attaching the encrypted PICCData parameter plus SDMMAC.

Table 32. Writing the NDEF File content

Step	Command	Data Message
1	File Nr	= 0x02
2	IV	= 00000000000000000000000000000000
3	Data (NDEF header + URL in hex)	= 0x005AD10156550068747470733A2F2F7777772E6E78702E6 36F6D2F696E6465782E68746D6C3F656E633D30303030303 03 030266D61633D3030303030303030303030303030303030
4	Length	= 0x5F0000
5	Offset	= 0x000000

Table 32. Writing the NDEF File content...continued

Step	Command	Data Message
6	C-APDU Cmd.WriteData	> 0x3D0200000005F0000005AD10156550068747470733A2F2F77 77772E6E78702E636F6D2F696E6465782E68746D6C3F656E 633D3030303030303030303030
7	R-APDU	< 0xAF (Additional Frame)
8	C-APDU	> 0xAF30266 D61633D30
9	R-APDU	< 00 (SUCCESS)

4.2.8.2.2 Changing the file settings of the NDEF File

In the following steps, the file settings of the NDEF file will be adapted, to configure the mirroring options.

As already outlined in, the following URL with the defined mirroring placeholders shall be realized:

<https://www.nxp.com/index.html?enc=00000000000000000000000000000000&mac=0000000000000000>

File Data Content:

[illegible]

NDEF File starting bytes: 005AD101565500

[illegible]

Authentication was executed prior using Key 0x03 with Key Value 0x11112222333344445555666677778888.

The resulting session key value is 0x4CE0100738A20BD1E23835F03D7B711F.

Table 33. Changing the file settings of the NDEF File

Step	Command	Data Message
1	File.Nr	= 0x02
2	FileOption	= 0x40 (SDM enabled, CommMode Plain)
3	New Access Rights	= 0x23E2 (R = Free, W = 0x02, RW = 0x02, Change = 0x03)
4	NrAdditional Access Rights	= n.a.
5	Additional Access Rights	= n.a.
6	SDMOptions	= 0xC1 (VUID mirroring enabled, SDMReadCtr enabled, ASCII encoding mode)

Table 33. Changing the file settings of the NDEF File ...continued

Step	Command	Data Message
7	SDMAccessRights	= 0xF333 SDMMetaReadKey = 0x3 (Encrypted PICCData mirroring) SDMFileReadKey = 0x3 SDMCtrRetKey = 0x3
8	VCUID Offset	= n.a.
9	SDMReadCtr Offset	= n.a.
10	PICCData Offset	= 0x2A0000
11	SDMMACInput Offset	= 0x4F0000 (means MAC calculation of the empty string)
12	SDMENC Offset	= n.a.
13	SDMENC Length	= n.a.
14	SDMMAC Offset	= 0x4F0000
15	SDMReadCtr Limit	= n.a.
16	TMC Limit	= n.a.
17	CRC32 Input = (Cmd CmdHeader CmdData)	= 5F024023E2C1F3332A00004F00004F0000
18	CRC32	= 515EED42
19	Enc(KSesAuthKey, CmdData CRC32 Padding)	= Enc(4023E2C1F3332A00004F00004F0000515EED420000000000000000000000000000)
20	Enc Output	= EAA7843F00C635A775316EF3592451E9AB234E59CF8929FCAF81CA089F9AAC3B
21	C-APDU Cmd.ChangeFileSettings	> 5F02EAA7843F00C635A775316EF3592451E9AB234E59CF8929FCAF81CA089F9AAC3B
22	R-APDU	< 001ADDBE6F338E0116
23	Received MAC	= 1ADDBE6F338E0116
24	IV	= AB234E59CF8929FCAF81CA089F9AAC3B
25	Calculated MAC = CMAC(KSesAuthKey, 00)	= 1ADDBE6F338E0116C1BEF35C35083116 = 1ADDBE6F338E0116
26	Received MAC? = Calculated MAC	= 1ADDBE6F338E0116 == 1ADDBE6F338E0116 (SUCCESS)

4.2.8.2.3 NDEF Validation: PICCData decryption and CMAC Calculation

SDMFileReadKey Nr = 0x3

SDMFileReadKey value = 0x11112222333344445555666677778888

Data content that was read from the file using SDM =

0x005AD10156550068747470733A2F27777772E6E78702E636F6D2F696E6465782E
68746D6C3F656E633D333932463637323937323437363737444236383238423536393
8453930303536266D61633D30373637303343394535423233333042

ASCII Text of the NDEF URL =

<https://www.nxp.com/index.html?enc=392F67297247677DB6828B5698E90056&mac=076703C9E5B2330B>

Table 34. Decryption of PICCData

Step	Command	Data Message
1	Encrypted PICCData	= 392F67297247677DB6828B5698E90056
2	SDMMetaReadKey = Key 0x03	= 11112222333344445555666677778888
3	IV	= 00000000000000000000000000000000
4	Dec($K_{SDMMetaReadKey}$, PICCENCDData)	= C704DE5F1EACC0403D0000DA5CF60941 C700C988631401060A0000B47D7F39D8
5	PICCDataTag	= C7
6	UID	= 00C98863140106
7	SDMReadCtr	= 0A0000
8	Random padding	= B47D7F39D8
9	PICCDataTag [bit]	= 1100 0111
10	PICCDataTag - UID mirroring [bit7]	= 1 (UID mirroring enabled)
11	PICCDataTag - SDMReadCtr mirroring [bit6]	= 1 (SDMReadCtr mirroring enabled)
12	PICCDataTag - UID Length [bit3-0]	= 111b = 7d (7 byte UID)

After the decryption of the PICC Data, UID and SDMReadCounter are known. With these values, the session key generation can be started and the verification of the SDMMAC can be done.

Table 35. SDM Session Key Generation

Step	Command	Data
1	Is UID mirrored?	= YES (it must be included in SV calculation)
2	Is SDMReadCtr mirrored?	= YES (it must be included in SV calculation)
3	UID	= 00C98863140106
4	SDMReadCtr	= 0A0000
5	$K_{SDMFileRead}$	= 11112222333344445555666677778888
6	$SV1 = C33C\ 0001\ 0080\ [UID]\ [SDMReadCtr]\ [ZeroPadding]$ ^[1]	= C33C0001008000C988631401060A0000
7	$SV2 = 3CC3\ 0001\ 0080\ [UID]\ [SDMReadCtr]\ [ZeroPadding]$	= 3CC30001008000C988631401060A0000
8	$K_{SesSDMFileReadENC} = MAC(K_{SDMFileRead}, SV1)$	= 760841CBFFF39A6236519844199F0DAD
9	$K_{SesSDMFileReadMAC} = MAC(K_{SDMFileRead}, SV2)$	= B79E240D2A03453488CF2623FA6D7047

[1] In case of encrypting file data - PICCENCDData, mirroring of UID and SDMReadCtr is mandatory. Therefore, both are always included in SV1 calculation.

Table 36. SDMMAC Validation

Step	Command	Data
1	Mirrored CMAC in URL	= 0x076703C9E5B2330B
2	$K_{SesSDMFileReadMAC}$	= 0xB79E240D2A03453488CF2623FA6D7047

Table 36. SDMMAC Validation...continued

Step	Command		Data
3	SDMMACInput Offset	=	0x4F0000
4	SDMMAC Offset	=	0x4F0000
5	SDMMACInput Offset == SDMMAC Offset	=	YES, therefore SDMMAC calculation over the empty string
6	IV	=	0x00000000000000000000000000000000
7	SDMMAC Input	=	"" (empty string)
8	Calculated SDMMAC	=	0x076703C9E5B2330B
9	Calculated SDMMAC == Mirrored SDMMAC	=	True (SUCCESS)

4.2.8.3 Example: SDM Mirroring with encrypted File Data (SDMEncFileData)

The following example shows how to write a weblink (URL) into the NDEF file.

The URL that is used here is <https://www.nxp.com/index.html>.

Additionally the SDM shall be enabled and effectively used for this file.

In this example, the data of the file will be mirrored encrypted and attached to the URL. The encrypted file data is the `SDMEncFileData`.

When using SDM, the file data is mirrored encrypted and can be read with the specified key for SDMFileRead access.

When using standard secure messaging, after an authentication has been established, the data can be transferred as per the secure messaging rules (plain, MACed or encrypted) and as per the communication mode that is set for the file.

Additionally to the encrypted File Data, the SDMMAC, SDMReadCounter and VCUID shall be mirrored. Their mirror offsets within NDEF can be individually chosen.

- VCUID
- SDM Read Counter
- SDM Enc File Data
- SDMMAC

Applying the mapping, we want to have a URL structure like below:

[illegible]

With this URL we're linking to the target website, and additionally attaching mirroring placeholders for the UID (?uid=), for the SDM Read Counter (&ctr=), encrypted file data (&enc_data=) and additionally for the SDM MAC (&mac=). The placeholder for the SDMMAC is initially filled with zeros, but will be updated on each tap of the IC automatically. The enc_data placeholder can be set depending on whatever the user wants to write into the file and shall be mirrored together with the URL. Once reading out the file content, the parameter will reflect the updated values.

Now that the mirroring parameters and URL structure are defined, two steps need to be done for configuration completion:

- Writing of the NDEF URL to the NDEF file
- Adapting the file settings including the SDM mirroring configuration for the NDEF file

To enable the mentioned settings, the `ChangeFileSettings` command needs to be used. For this command, the `SDMOptions` need to be set to `0x40`.

4.2.8.3.1 Writing the NDEF File content

The following example shows how to write a weblink (URL) into the NDEF file.

[illegible]

The file data that shall be inserted to the file as enc_data is:

0x4444444455555555666666667777777788888888999999997777777766666666

Therefore the data to be writing together with the NDEF header is:

0x005AD10156550068747470733a2f2f7777772e6e78702e636f6d2f696e6465782e68746
d6c3f7569643d303030303030303030303030266374723d303030303026656e635
f646174613d3434343434343435353535353535353636363636363637373737373
737383838383838383839393939393939393737373737373736363636363636266d
61633d30303030303030303030303030303030

Table 37. Writing the NDEF File content

Step	Command	Data Message
1	File Nr	= 0x02
2	IV	= 00000000000000000000000000000000
3	Data (NDEF header + URL in hex)	= 0x005AD10156550068747470733a2f2f777772e6e78702e636 f6d2f696e6465782e68746d6c3f7569643d30303030303030303 03030303030266374723d30303030303026656e635f646174613 d343434343434343435353535353535353535363636363636363 737373737373737373838383838383838383839393939393939373 737373737373737363636363636363636d61633d303030303030 0303030303030303030303030
4	Length	= 0xA20000
5	Offset	= 0x000000
6	C-APDU Cmd.WriteData	> 0x3D02000000A20000005AD10156550068747470733A2F2F77 77772E6E78702E636F6D2F696E6465782E68746D6C3F75696 43D3030303030303030303030
7	R-APDU	< 0xAF (Additional Frame)
8	C-APDU	> 0xAF30303030266374723D303030303026656E635F64617 4613D3434343434343434343535353535353535363636363636 363737373737373737373838
9	R-APDU	< 0xAF (Additional Frame)
10	C-APDU	> 0xAF3838383838383939393939393939393937373737373736 36363636363636266D61633D30303030303030303030303030 303030
11	R-APDU	< 00 (SUCCESS)

4.2.8.3.2 Changing the file settings of the NDEF File

In the following steps, the file settings of the NDEF file will be adapted, to configure the mirroring options.

As already outlined in, the following URL with the defined mirroring placeholders shall be realized:

https://www.nxp.com/index.html?uid=00000000000000&ctr=000000_&enc_data=444444445555555566666666777777777788888888999999997777777766666666&mac=0000000000000000

Authentication was executed prior using Key 0x03 with Key Value 0x11112222333344445555666677778888.

The resulting session key value is 0xDE60AFCD0FEDEBB1A3987DB944938979.

Note: Important to consider when mirroring the *SDMEncFileData*, the *SDMMAC* calculation needs to include the complete mirrored file data, meaning the *SDMMAC* Offset needs to be set accordingly, to start with the *SDMEncFileData* Offset or earlier.

Note: When writing a certain length of data (*SDMEncFileData*), the length parameter needs to cover the actual length in bytes times 2 (for matching the ASCII format length). In this scenario a length of 0x400000 is specified, actually covering 0x200000 bytes in HEX format (32 bytes).

Additionally to consider for ASCII encoding, only the first half of the placeholder is used for storing the plain data, the second half will be ignored for constructing the returned data when reading with SDM. For example, if targeting to encrypt 2 AES blocks, i.e. 32 bytes, a placeholder of 64 bytes is reserved via *SDMENCOffset* and *SDMENCLength*. The first 32 bytes will hold the plaintext, and the next 32 bytes are ignored when reading with Secure Dynamic Messaging.

In our scenario this means that actually only the first 16 bytes of plaintext will be mirrored and encrypted, and attached in ASCII format to the URL, as we are only reserving 32 bytes of mirroring space using a length of 0x200000. Therefore only the data 0x44444444555555556666666677777777 will be encrypted and mirrored.

Table 38. Changing the file settings of the NDEF File

Step	Command	Data Message
1	File Nr	= 0x02
2	FileOption	= 0x40 (SDM enabled, CommMode Plain)
3	New Access Rights	= 0x23E2 (R = Free, W = 0x02, RW = 0x02, Change = 0x03)
4	NrAdditional Access Rights	= n.a.
5	Additional Access Rights	= n.a.
6	SDMOptions	= 0xD1 (VCUID mirroring enabled, SDMReadCtr mirroring enabled, SDMEncFileData mirroring enabled, ASCII encoding mode)
7	SDMAccessRights	0xF3E3 = SDMMetaReadKey = 0xE (Plain PICCData mirroring) SDMFileReadKey = 0x3 SDMCtrRetKey = 0x3
8	VCUID Offset	= 0x2A0000
9	SDMReadCtr Offset	= 0x3D0000
10	PICCData Offset	= n.a.
11	SDMMACInput Offset	= 0x4D0000 (means MAC calculation of the complete <i>SDMENCFFile</i> Data Input string)
12	SDMENC Offset	= 0x4D0000
13	SDMENC Length	= 0x400000
14	SDMMAC Offset	= 0x920000
15	SDMReadCtr Limit	= n.a.
16	TMC Limit	= n.a.
17	CRC32 Input = (Cmd CmdHeader Cmd Data)	= 5F024023E2D1F3E32A00003D00004D00004D0000400000920000
18	CRC32	= EFB22E16

Table 38. Changing the file settings of the NDEF File ...continued

Step	Command		Data Message
19	Enc(KSesAuthKey, CmdData CRC32 Padding)	=	Enc(4023E2D1F3E32A00003D00004D00004D000400000920000EFB22E1600000000)
20	Enc Output	=	0920B4E7A5ECC61BA04E2B2A950F763642BF7067C08D6BC7BCA5B26C845366B4
21	C-APDU Cmd.ChangeFileSettings	>	5F020920B4E7A5ECC61BA04E2B2A950F763642BF7067C08D6BC7BCA5B26C845366B4
22	R-APDU	<	00795292FAF00E987C
23	Received MAC	=	795292FAF00E987C
24	IV	=	42BF7067C08D6BC7BCA5B26C845366B4
25	Calculated MAC = CMAC (KSesAuthKey, 00)	=	795292FAF00E987C3604A79BA5BF4132 = 795292FAF00E987C
26	Received MAC ?= Calculated MAC	=	795292FAF00E987C == 795292FAF00E987C (SUCCESS)

4.2.8.3.3 NDEF Validation: File Data decryption and CMAC Calculation

SDMFileReadKey Nr = 0x3

SDMFileReadKey value = 0x11112222333344445555666677778888

Data content that was read from the file using SDM =

0x005AD10156550068747470733A2F2F7777772E6E78702E636F6D2F696E6465782
E68746D6C3F7569643D3030433938383633313430313036266374723D3030303030
4426656E635F646174613D3733374331343331383230413142303039343131373743
453230414543464335303742303031374633373043383346334630343444414244314
2333639464245266D61633D42353131303332324343393037383044000000000000
00

ASCII Text of the NDEF URL =

https://www.nxp.com/index.html?uid=00C98863140106&ctr=00000D&enc_data=737C1431820A1B00941177CE20AECFC507B0017F370C83F3F044DABD1B369FBE&mac=B5110322CC90780D

As first step, before the decryption of the SDMEncFileData can start, the SDM session keys need to be calculated, afterwards the SDMEncFileData can be decrypted, and the SDMMAC can be validated.

Table 39. SDM Session Key Generation

Step	Command		Data
1	Is UID mirrored?	=	YES (it must be included in SV calculation)
2	Is SDMReadCtr mirrored?	=	YES (it must be included in SV calculation)
3	UID	=	00C98863140106
4	SDMReadCtr	=	0D0000
5	KSDMFileRead	=	11112222333344445555666677778888
6	SV1 = C33C 0001 0080 [UID] [SDMReadCtr] [ZeroPadding] ^[1]	=	C33C0001008000C988631401060D0000

Table 39. SDM Session Key Generation...continued

Step	Command		Data
7	SV2 = 3CC3 0001 0080 [UID] [SDMReadCtr] [ZeroPadding]	=	3CC30001008000C988631401060D0000
8	$K_{\text{SesSDMFileReadENC}} = \text{MAC}(K_{\text{SDMFileRead}}, \text{SV1})$	=	8C3D4C81D7890D5D0EB50EC94C3D3DF3
9	$K_{\text{SesSDMFileReadMAC}} = \text{MAC}(K_{\text{SDMFileRead}}, \text{SV2})$	=	F08A9DA14A00F7BB3AE849CCAF88DA1E

[1] In case of encrypting file data - PICCENCDData, mirroring of UID and SDMReadCtr is mandatory. Therefore, both are always included in SV1 calculation.

Table 40. Decryption of SDMEncFileData

Step	Command		Data Message
1	Encrypted File Data (SDMEncFileData)	=	0x737C1431820A1B00941177CE20AECFC507B0017F370C83F3F044DABD1B369FBE
2	SesSDMFileReadENC	=	0x8C3D4C81D7890D5D0EB50EC94C3D3DF3
3	$\text{IV} = \text{Enc}(K_{\text{SesSDMFileReadENC}}, \text{SDMReadCtr} \parallel 0x00000000000000000000000000000000)$	=	$\text{Enc}(0D000000000000000000000000000000) = 6C4E2A133D657EA88D7C340ADBF639F5$
4	Decrypted File Data in ASCII Format $\text{Dec}(K_{\text{SesSDMFileReadENC}}, \text{SDMEncFileData})$	=	3434343434343434353535353535353536363636363636363737373737373737
5	Decrypted File Data in HEX Format	=	0x44444444555555556666666677777777
6	Decrypted File Length	=	0x200000
7	SDMReadCtr	=	0A0000
8	Random padding	=	B47D7F39D8
9	PICCCDataTag [bit]	=	1100 0111
10	PICCCDataTag - UID mirroring [bit7]	=	1 (UID mirroring enabled)
11	PICCCDataTag - SDMReadCtr mirroring [bit6]	=	1 (SDMReadCtr mirroring enabled)
12	PICCCDataTag - UID Length [bit3-0]	=	111b = 7d (7 byte UID)

After the decryption of the SDMEncFileData, the plaintext input is known and the verification of the CMAC can be done.

Table 41. SDMMAC Validation

Step	Command		Data
1	Mirrored CMAC in URL	=	0xB5110322CC90780D
2	$K_{\text{SesSDMFileReadMAC}}$	=	0xF08A9DA14A00F7BB3AE849CCAF88DA1E
3	SDMMACInput Offset	=	0x4D0000
4	SDMMAC Offset	=	0x920000
5	SDMMACInput Offset == SDMMAC Offset	=	NO, SDMMAC calculation covers the file data
6	IV	=	0x00000000000000000000000000000000
7	SDMMAC Input (ASCII)	=	737C1431820A1B00941177CE20AECFC507B0017F370C83F3F044DABD1B369FBE&mac=
8	SDMMAC Input (HEX)	=	0x37333743313433313832304131423030393431313737434532304145434643353037423030313746333730433833463346303434444142443142333639464245266d61633d

Table 41. SDMMAC Validation...continued

Step	Command		Data
9	Calculated SDMMAC	=	0x11B5DB1147038A221DCCA7905578FD0D (Full) = 0xB5110322CC90780D (Truncated)
10	Calculated SDMMAC == Mirrored SDMMAC	=	True (SUCCESS)

4.2.9 SDM Counter Retrieval

Once Secure Dynamic Messaging was enabled for a file, the `SDMReadCounter` is also maintained for the file, counting every `ReadData` and `ISORReadBinary` commands that have been used for accessing the file data (exceptions and special conditions read in [\[1\]](#).)

An easy way to read out the value of the `SDMReadCounter` is the retrieval via the mirroring as part of the `PICCCData`, or it can alternatively also be retrieved via `GetFileCounters` command, as shown in [Section 4.2.9.1](#).

4.2.9.1 Example: Retrieving the `SDMReadCtr` using `GetFileCounters` command

This example shows the usage of the `GetFileCounters` command and how the currently used `SDMReadCtr` can be retrieved from a file.

Precondition for the execution of this command is the authentication with the `SesSDMCTRRetKey`.

In this example the `SesSDMCTRRetKey` was set during the `ChangeFileSettings` command to application key 0x03 with key value 0x11112222333344445555666677778888. The authentication has been performed with `AuthenticateAES` and the resulting session key has the value 0xDE19D4AD9B857C6BED9D6842643A67AE.

Table 42. `SDMReadCtr` Retrieval using `GetFileCounters` command

Step	Command		Data
1	File Nr	=	0x02
2	<code>K_{SesSDMCTRRetKey}</code>	=	0xDE19D4AD9B857C6BED9D6842643A67AE
3	MAC Calculation = <code>CMAC(K_{SesSDMCTRRetKey}, F602)</code>	=	<code>CMAC(F602)</code> = 0x388E0C335E7C310611EF8D4C110F824A
4	C-APDU <code>GetFileCounters</code>	=	0xF602
5	R-APDU	=	0x00A3D7E46616CD6E4432EBAD96EA7149F7
6	Encrypted <code>ResponseData</code>	=	0xA3D7E46616CD6E4432EBAD96EA7149F7
7	IV	=	0x388E0C335E7C310611EF8D4C110F824A
8	<code>Dec(K_{SesSDMCTRRetKey}, EncResponseData)</code>	=	0x0D00000000824DA3F200000000000000 (<code>ReadCtr</code> RFU <code>CRC32</code> Padding)
9	<code>ReadCtr</code>	=	0x0D0000
10	RFU	=	0x0000
11	<code>CRC32 = CRC32(ReadCtr RFU ResponseCode)</code>	=	0x824DA3F2
12	Padding	=	0x0000000000000000
13	Response Code	=	0x00 (SUCCESS)

5 PICC Memory and Configuration management

The complete user memory is available for application and file creation. All PICC level keys and configuration data are located outside the user memory, certificates and generally files are counting towards the user memory.

All commands which impact the memory structure (e.g. through creation or deletion of application or files) activate an automatic backup mechanism that protects the application and file structure from being corrupted. The backup mechanism is called tearing mechanism and ensures that the memory stays intact.

The free memory of the card can be determined using `Cmd.FreeMem`.

`Cmd.Format` can be used for formatting and releasing user memory. If executing this command while the PICC is selected, `Cmd.Format` releases all allocated user memory. If a single application is selected, the command is rejected if this application is not a delegated application. If a delegated application is selected, the command releases all allocated file memory in the application. This means that all files are deleted and the memory can be reused, but the application configuration settings and the application keys remain on the PICC.

5.1 MIFARE DUOX Memory Organization

The memory which can be used for application and file creation in MIFARE DUOX is allocated in blocks of 32 bytes. Although the naming of the individual product variants of MIFARE DUOX is telling storage size is 2 kB, 4 kB, 8 kB and 16 kB, the size of the really available memory is defined in [Table 43].

Table 43. MIFARE DUOX Supported Memory Configurations

Memory configuration	Size in bytes	Size in blocks
2 kB	2560	0x0050
4 kB	5120	0x00A0
8 kB	8192	0x0100
16 kB	16384	0x0200

5.2 Memory consumption due to Key Usage

As already stated, PICC keys are stored outside the user memory, so they don't impact the memory consumption at all. However, application keys are located inside the user memory and therefore also require some dedicated memory. Memory allocation is done internally in 32-byte blocks.

Which application level keys are available and which commands can be executed with them will be discussed in detail in [Section 6](#).

Symmetric Keys:

Basically can be differentiated between two cases for application key memory consumption:

- If the application does not hold any keys
Required memory = 0 blocks
Precondition: `Cmd.CreateApplication.KeySet2` or `Cmd.CreateDelegatedApplication.Bit 3-0` are set to 0x00
- If the application holds keys (normal application)
 $\text{NoKeySets} * (1 + (\text{NoAppKeys} > 3 ? 1 : 0) + (\text{NoAppKeys} > 11 ? 1 : 0) + \text{ceil}(\text{NoKeyBlocks} / 4))$ memory blocks

Depending on the key size, also the number of key blocks differs:

- If MaxKeySize is 16 byte
 $\text{NoKeyBlocks} = \text{NoAppKeys} \times 2$
- If MaxKeySize is 24 byte
 $\text{NoKeyBlocks} = \text{NoAppKeys} \times 3$

The explanation of the terms used in this calculation is like the following:

- NoKeySets = The number of key sets that is present in an application
- NoKeyBlocks = The number of used 8 byte blocks per key (2 for AES128 keys, and 4 for AES256 keys)
- NoAppKeys = The number of application keys per key set

Asymmetric Keys:

The memory for asymmetric keys is allocated **at their creation** (different from symmetric keys which are jointly created with an application) and is defined as follows:

- KeyID.ECCPrivateKey: **3 blocks**
- KeyID.CARootKey: **3 blocks**

Additionally, an application will allocate memory for the following item if used

- Delegated applications enabling KeyID.AppMasterTmpKey: **2 blocks**

6 Symmetric Key Management

MIFARE DUOX supports a set of PICC level keys (see [Section 6.3](#)) and multiple sets of application level keys (see Application Level Keys). The security capabilities of MIFARE DUOX are related to these keys. MIFARE DUOX grants access to certain functionality depending on its configuration and the knowledge of certain keys. It is not possible to read out any of these PICC level or application level keys, as the MIFARE DUOX key management functionality ensures key confidentiality and integrity.

6.1 Key Types

MIFARE DUOX supports two different AES key lengths :

- AES-128, 16 byte key length
- AES-256, 32 byte key length

The key type to be used in an application can be defined at application creation (Bit 7-6 of KeySet2).

6.2 Key Versioning

MIFARE DUOX supports the versioning of keys by relating each key with a 1 byte key version. The version of any addressable key can be read using [Cmd.GetKeyVersion](#). The key versions can be used to increase the level of security e.g. multiple keys (distinct by key version) for the same application.

The version number for KeyType.AES keys is stored in a separate byte next to the key. Using [Cmd.ChangeKey](#) or [Cmd.ChangeKeyEV2](#), the key version number needs to be specified.

6.3 PICC Level Keys

Table 44. Keys at PICC Level and their Usages

Key Name	Nr	Key Type	Key Usage	Key Change
KeyID.PICCMasterKey	0x00	KeyType.AES	<ul style="list-style-type: none"> • Changing of KeyID.PICCMaster Key, all DAM Keys, and KeyID.VCConfigurationKey (if allowed by other settings) • Changing of any PICC configuration using Cmd.Set Configuration • Formatting the PICC (if not disabled) • Changing the PICC KeySet1 (if allowed by KeySet1) • Creating/Deleting an application (if required by KeySet1) • Executing the Cmd.GetApplication IDs, Cmd.GetKeySettings, and more (if required by KeySet1) 	<ul style="list-style-type: none"> • Use Cmd.ChangeKey or Cmd.ChangeKeyEV2 • Only after authentication with KeyID.PICCMaster Key • Key can be made unchangeable by Key Set1

Table 44. Keys at PICC Level and their Usages...continued

Key Name	Nr	Key Type	Key Usage	Key Change
KeyID.PICCDAMAuthKey	0x10	KeyType.AES	<ul style="list-style-type: none"> Authentication with KeyID. PICCDAMAuthKey is required for creating a delegated application 	<ul style="list-style-type: none"> Use Cmd.ChangeKey or Cmd.ChangeKeyEV2 Only after authentication with KeyID.PICCMaster Key
KeyID.PICCDAMMACKey	0x11		<ul style="list-style-type: none"> Calculating the cryptogram (CMAC calculation) on Cmd.Create DelegatedApplication command data 	
KeyID.PICCDAMENCKey	0x12		<ul style="list-style-type: none"> Encrypting the KeyID. PICCDAMDefaultKey 	
KeyID.NXPDAAuthKey	0x18	KeyType.AES	<ul style="list-style-type: none"> Authentication with KeyID. NXPDAAuthKey is required for creating a delegated application via the NXP AppXplorer 	<ul style="list-style-type: none"> These keys cannot be changed, but are set by NXP during IC production
KeyID.NXPDAAMACKey	0x19		<ul style="list-style-type: none"> Calculating the cryptogram (CMAC calculation) on Cmd.Create DelegatedApplication command data via the NXP AppXplorer 	
KeyID.NXPDAENCKey	0x1A		<ul style="list-style-type: none"> Encrypting the KeyID. PICCDAMDefaultKey via the NXP AppXplorer 	
KeyID.VCConfigurationKey	0x20	KeyType.AES	<ul style="list-style-type: none"> These keys are associated with card management (VC) and proximity checking (PC) An application can have its own proximity key and so overrule these PICC level VC and PC keys 	<ul style="list-style-type: none"> Use Cmd.ChangeKey or Cmd.ChangeKeyEV2 Only after authentication with KeyID.PICCMaster Key or KeyID.VCConfigurationKey Changing of KeyID.VCConfigurationKey requires authentication with KeyID.PICCMaster Key
KeyID.VCProximityKey	0x21			
KeyID.ICUpgradeKey	0x30	KeyType.AES	<ul style="list-style-type: none"> Used during the AuthenticatePDC command Associated to post-delivery configuration options of MIFARE DUOX 	<ul style="list-style-type: none"> This key cannot be changed
KeyID.PICCAppDefaultKey	-	-	<ul style="list-style-type: none"> Holds the initialization value and version of each key (of a key set) used in conventional application creation This key is non-addressable 	<ul style="list-style-type: none"> Can be changed via Cmd.SetConfiguration Only after authentication with KeyID.PICCMaster Key

6.4 Application Level Keys

Each application has its own keys (see [Section 6.4.2](#)) which are addressed as soon as the application is selected. The regular `KeyID.AppKeys` are grouped into key sets, and additional to these, each application can also have its own `Proximity Check Key`. As soon as an application gets created, the keys from the first key set are initialized to default key values and default key versions (of `KeyID.PICCAppDefaultKey` for conventional applications and `KeyID.AppDAMDefaultKey` for delegated applications). In order to be able to roll to another keyset, an initialization of the keyset (that should be rolled to) needs to be done first.

6.4.1 Application Key Sets

MIFARE DUOX supports the usage of multiple keys sets. Each application can have up to 16 key sets, each having same number of keys (up to 14). Key set parameters (number of key sets, number of keys, max key size, key set version, and key set type) are defined at application creation and it's not possible to change it thereafter. If key set parameters are not provided at application creation, the application is created with single key set. An application can be also created without any key and therefore also without any key set.

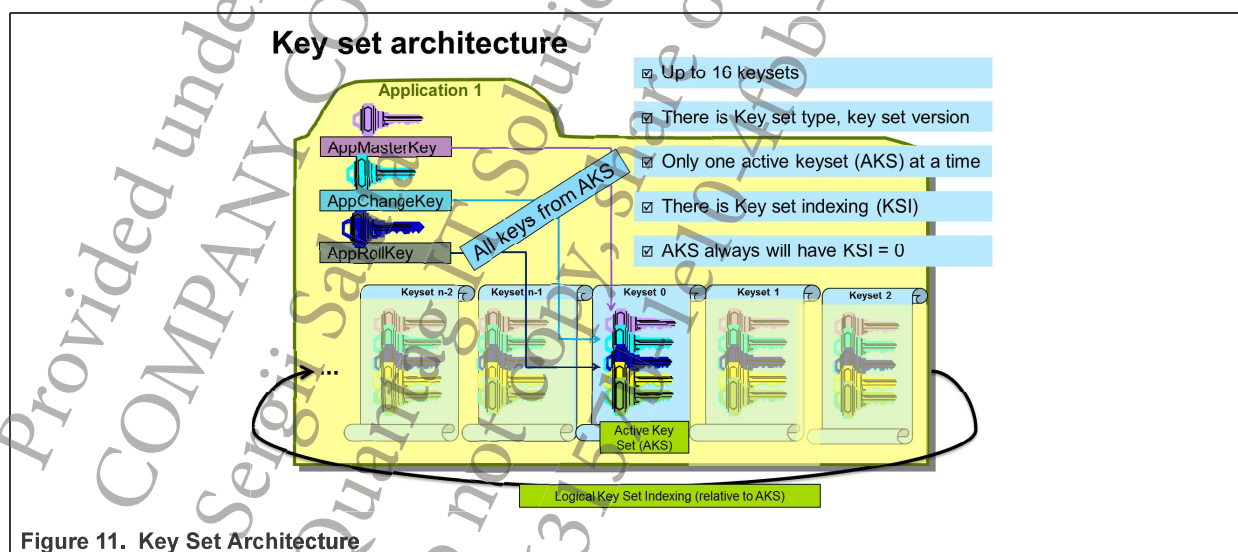
The multiple key sets feature increases system security through a secure key renewal policy. With the multiple key sets feature of MIFARE DUOX, rolling to a new key set (in a way to the new key value and key type) is possible by simply making an authentication using `KeyID.AppRollKey` (fulfilling other conditions - see [Section 6.4.1.6 "Roll Key Set"](#)).

6.4.1.1 Key Set Architecture

Only one key set within an application is active at a time and this key set is called AKS (Active key Set). All the keys necessary for authentication come from the AKS, that means that authentication is only possible by using a key from the active key set.

There is a so-called key set indexing (KSI) available, and the AKS is always characterized through the KSI having the value 0x00. This means that the AKS is addressed by the key set number 0x00, and all the other key set indices are relative to the AKS. When the AKS is updated to point to another key set, the address numbering is changed and updated relatively to the new AKS.

[Figure 11](#) depicts the described key set architecture graphically.



Each key set is characterized by some properties. These properties can be either defined at application level or at key set level (during key set personalization). Application level properties are shared between all key sets of one application.

Each key set has a key set version. This key set version is different to the key version. The version of all key sets can be read using the `Cmd.GetKeyVersion`, by setting the 6th bit to 1. This is important in order to find out which is the latest key set in use.

A key set also has a key set type. All keys from one key set have the same key type. Different key sets can be personalized to different key types.

6.4.1.2 Key Set Personalization

After the successful execution of `Cmd.CreateApplication` or `Cmd.CreateDelegatedApplication` the AKS is initialized at application level according to the key type given.

- All the keys of the AKS hold the value and version of `KeyID.PICCAAppDefaultKey` (for conventional application) or `KeyID.AppDAMDefaultKey` (for delegation application).
- All the keys have the key type provided in `KeySett.2`
- The AKS has the key set version provided in `AKSVersion`.

After the application level initialization, the key sets can be personalization in three steps:

- Step1: Initialize new key set with `Cmd.InitializeKeySet`
- Step2: Change keys (of key set) with `Cmd.ChangeKeyEV2`
- Step3: Finalize new key set (now key set is ready to be rolled) with `Cmd.FinalizeKeySet`

These steps are described in the following sections.

Example: Application creation with defined key set parameters

KeySett1 = 1F (Change key: 0x01, all configuration enabled)

KeySett2 = B5 (AES key type, Key set param present, 5 keys per key set)

KeySett3 = 01

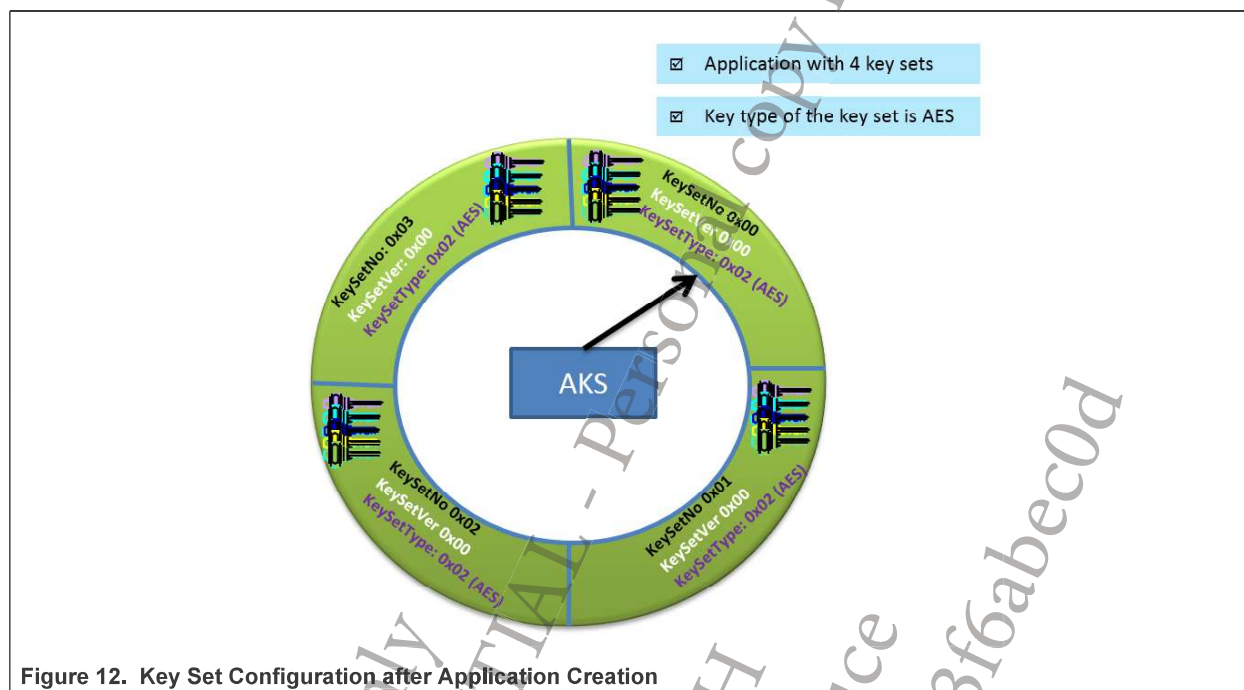
AKSVersion = 00

No of key Set = 04

Max Key Size = 10 (16 byte)

Roll key = 02

After application creation, the key sets will look like the ones which are visible in [Figure 12](#).



6.4.1.3 Initialize Key Set

`Cmd.InitializeKeySet` initializes all the keys of the targeted key set with the corresponding value and version of the AKS keys. If the targeted key set should have a different key type than the AKS's key type, the targeted key set's key value needs to be padded with zeros or truncated accordingly.

The command also sets the key set version of the targeted key set to 0x00 (which indicates that key set is not finalized yet) and the key type to the given key type. `Cmd.InitializeKeySet` requires authentication with `KeyID.AppChangeKey` depending on `KeySett1`. If 0xE is set at `KeySett1` as change key, authentication with `KeyID.AppMasterKey` is necessary. If 0xF is set i.e. it is frozen, nothing is possible with key sets or keys. Please note that the AKS can never be initialized. The rest of the key sets can be reinitialized at any time.

6.4.1.3.1 Example: `Cmd.InitializeKeySet` in EV2 Secure Messaging

Currently EV2 Secure Messaging is active (authentication was done with `KeyID.ChangeKey`, 0x01 in this case).

`KeyID.SesAuthENCKey` = 6A52C626425C39504E371607E7545D6C

`KeyID.SesAuthMACKey` = 1F28BEAD76521D73FFF6E693B521E239

Current `CmdCtr` = 0000 (LSB first)

`TI` = 2BF0463E

`CMD` = 56

`KeySetNo` = 0x01

`KeySetType` = 0x02 (AES Type)

Table 45. Example for Cmd.InitializeKeySet in EV2 Secure Messaging

Step	Command		Data Message
1	Data for calculating CMAC (Cmd CmdCtr TI KeySetNo KeySetType)	=	5600002BF0463E0102
2	IV for CMAC calculation	=	00000000000000000000000000000000
3	CMAC to be sent	=	CF78815F7A22066C
4	Cmd.InitializeKeySet C-APDU	>	5600002BF0463E0102CF78815F7A22066C
5	Updated CmdCtr	=	0100
6	R-APDU	<	006CAA7E76675DD87C
7	Data for calculating CMAC on response (RC CmdCtr TI)	=	6CAA7E76675DD87C
8	CMAC to be received	=	6CAA7E76675DD87C
9	PCD compares calculated CMAC and received CMAC	=	6CAA7E76675DD87C ? 6CAA7E76675DD87C

6.4.1.4 Change Keys of a Key Set

Once a key set is initialized to the default value and version of corresponding key value and version from AKS, each of the keys (of that targeted key set) should be changed one after another.

Cmd.ChangeKeyEV2 is used for this key changes. With Cmd.ChangeKeyEV2, any key from any key set can be changed, including the AKS (in that case parameter KeySetNo needs to be 0x00).

Changing keys from any key set (including AKS) requires an authentication with KeyID.AppChangeKey or KeyID.AppMasterKey depending KeySet1. If 0xE is set in KeySet1:

- While changing a key from the AKS (KeySetNo = 0x00), authentication with targeted AppKey (from AKS) is required
- While changing a key from another key set (KeySetNo != 0x00), authentication with equivalent key (same key number from AKS) is required

When changing a key, the key number which needs to be changed can never be the key number used for the currently authenticated session:

Key number to be changed ≠ Key number of currently authenticated session

6.4.1.4.1 Example: Cmd.ChangeKey in EV2 Secure Messaging

In this example, the key number 0 of key set 1 will be changed to a new key. The Cmd.ChangeKey is only demonstrated here and should be repeated until all keys are changed. Currently EV2 SM is active (authentication is done with KeyID.AppChangeKey, KeyNo 0x01 in this case)

KeyID.SesAuthENCKey = 3D14286D998BF62B6B82C8A42DC33E3F

KeyID.SesAuthMACKey = 6C1CF8AAEBB0CDCDF79E1A1B8BEA15BD

Current CmdCtr = 0100 (LSB first)

TI = 6F9E6308

Table 46. Example for Cmd.ChangeKey in EV2 Secure Messaging

Step	Command	Data Message
1	Previous key value of key no 0x00	= 00112233445566778899AABBCCDDEEFF
2	Value of new key number 0x01	= A0A1A2A3A4A5A6A7A8A9AAABACADAEAF
3	Version for new key (intended)	= 03
4	Previous key XOR New Key	= A0B08090E0F0C0D02030001060704050
5	Cmd.ChangeKey	= C401
6	CRC32 calculated on new key (A0A1A2A3A4A5A6A7A8A9AAABACADAEAF)	= 90DBDA4D
7	Previous key XOR new key version of new key CRC32 for new key padding	= A0B08090E0F0C0D020300010607040500390 DBDA4D800000000000000000000000
8	Current IV	= 331655A5FE10ADF02CAB1A31E6BC1CAA
9	Encrypted data from step 7 with session key	= 218A2D144D64CE918A41C4712A2C9EF1E2 F868AA83E8DE76D4665E3DC4690177
10	Current IV	= E2F868AA83E8DE76D4665E3DC4690177
11	Cmd.ChangeKey (Cmd CmdCtr TI CmdHeader encrypted data from step 7 with session key)	> C401006F9E630801218A2D144D64CE918 A41C4712A2C9EF1E2F868AA83E8DE76 D4665E3DC4690177
12	R-APDU (Status CMAC)	< 00856F88AC2224C4CF
13	Calculate CMAC for status byte	= 856F88AC2224C4CF

6.4.1.5 Finalize Key Set

Cmd.FinalizeKeySet finalizes the key set by giving a version to the targeted key set. The provided version can be any value different than 0x00. At this moment, PICC doesn't check whether the given version is strictly greater than the AKS version.

It is not possible to finalize the currently selected AKS. Cmd.FinalizeKeySet requires authentication with KeyID.AppChangeKey depending on KeySet1. If 0xE is set at KeySet1 as change key, authentication with KeyID.AppMasterKey is necessary.

6.4.1.5.1 Example: Cmd.FinalizeKeySet in EV2 Secure Messaging

Currently EV2 SM is active (authentication is done with KeyID.AppChangeKey, KeyNo 0x01 in this case)

KeyID.SesAuthENCKey = 88C1FE0773508C090422BB1F29AC09D6

KeyID.SesAuthMACKey = E5860EA27916203FFCE2B6ABCBDE435D

Current CmdCtr = 0000 (LSB first)

TI = BC8DE63D

CMD = 57

KeySetNo = 0x01

New KeySetVer = 0x12

Table 47. Example for Cmd.FinalizeKeySet in EV2 Secure Messaging

Step	Command		Data Message
1	Data for calculating CMAC (Cmd CmdCtr TI KeySetNo KeySetType)	=	560000BC8DE63D0102
2	IV for CMAC calculation	=	00000000000000000000000000000000
3	CMAC to be sent	=	0D87572835D964BE
4	C-APDU	>	5601020D87572835D964BE
5	Updated CmdCtr**	=	0100
6	R-APDU	<	00465D6D4737EBE157
7	Data for calculating CMAC on response (RC CmdCtr TI)	=	000100BC8DE63D
8	CMAC to be received	=	465D6D4737EBE157
9	PCD compares calculated CMAC and received CMAC	=	465D6D4737EBE157? 465D6D4737EBE157

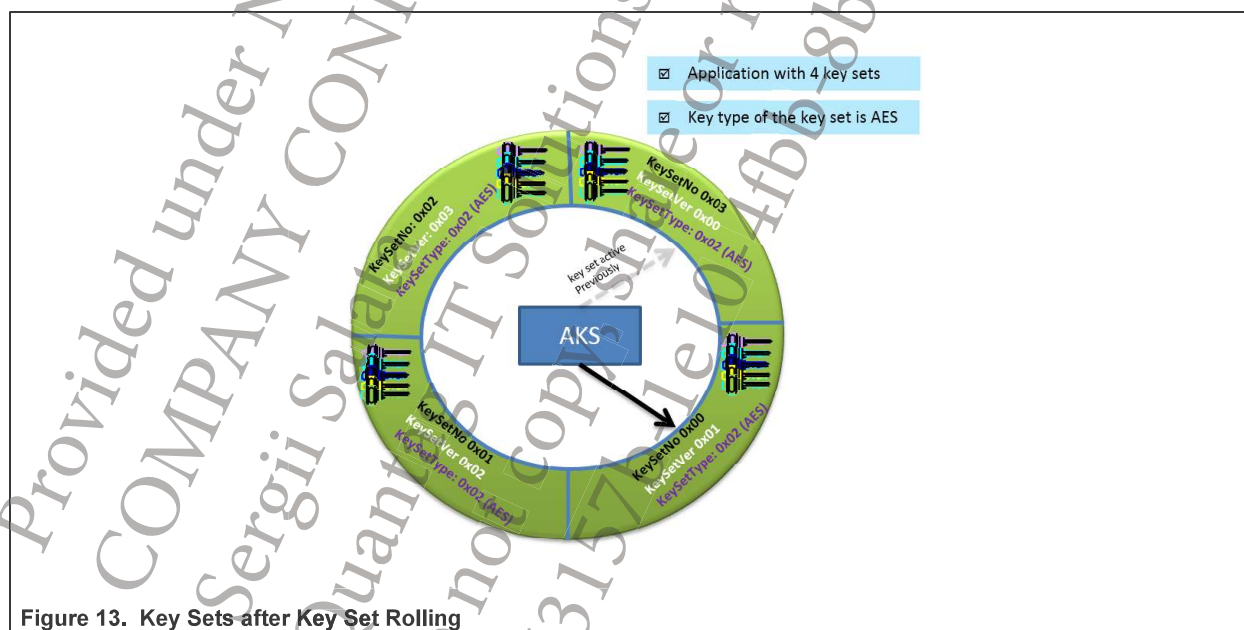
6.4.1.6 Roll Key Set

Once all the key sets are finalized, one can roll out to a new key set at any time in a fast and secure way. Rolling to a new key set requires nothing but just authentication with KeyID.AppRollKey (defined at application creation – it is not possible to change this key afterwards). Rolling is done with Cmd.RollKeySet.

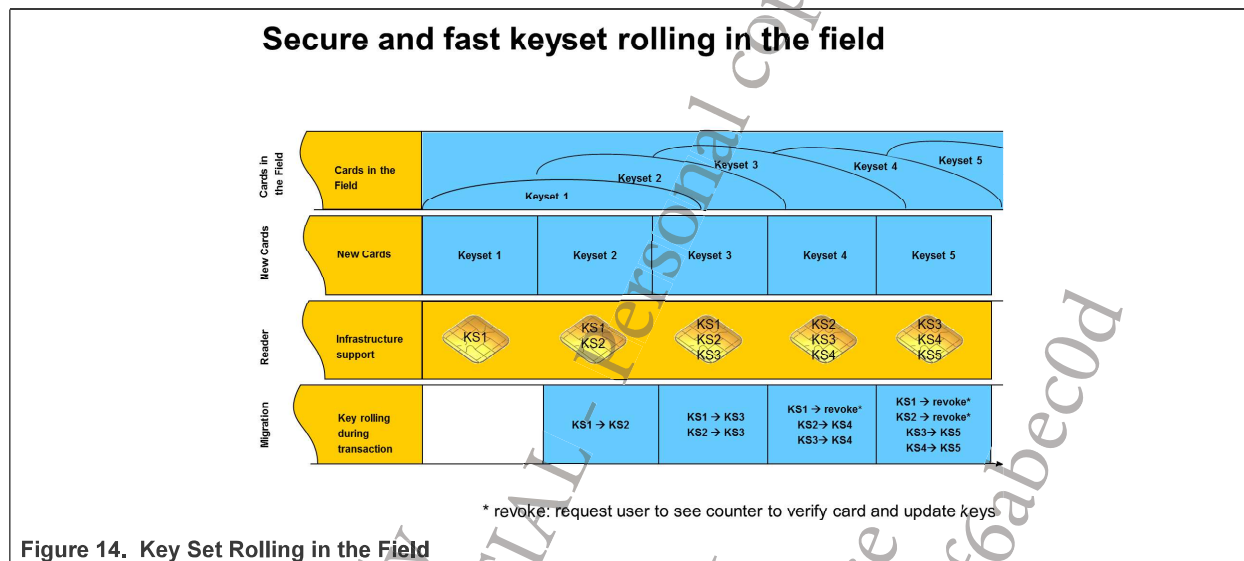
Rolling to a new key set is possible only if:

- Key set version of the targeted key set is strictly higher than the version of the AKS
- Key set type of the targeted key set is equal or higher than key type of the AKS (e.g. KeyType.AES-256 is stronger than KeyType.AES-128)

After key set rolling, the targeted key set becomes the new AKS and the key set numbers get updated as shown in [\[Figure 13\]](#).



Key Set Rolling can be performed easily and secure in the field. The only condition is, that one or more key sets beside the AKS have already been finalized. [Figure 14](#) gives an overview of key set rolling in the field.



6.4.1.6.1 Example: Cmd.RollKeySet in EV2 Secure Messaging

Currently EV2 SM is active (authentication is done with KeyID.AppRollKey, 0x02 in this case).

KeyID.SesAuthENCKey = 88C1FE0773508C090422BB1F29AC09D6

KeyID.SesAuthMACKey = E5860EA27916203FFCE2B6ABCBDE435D

Current CmdCtr = 0000 (LSB first)

TI = BC8DE63D

CMD = 57

KeySetNo = 0x01

New KeySetVer = 0x12

Table 48. CMAC calculation 2

Step	Command	Data Message
1	Data for calculating CMAC (Cmd CmdCtr TI KeySetNo KeySettype)	= 560000BC8DE63D0102
2	IV for CMAC calculation	00000000000000000000000000000000
3	CMAC to be sent	= 0D87572835D964BE
4	C-APDU	> 5601020D87572835D964BE
5	Updated CmdCtr**	= 0100
6	R-APDU	< 00465D6D4737EBE157
7	Data for calculating CMAC on response (RC CmdCtr TI)	000100BC8DE63D
8	CMAC to be received	= 465D6D4737EBE157
9	PCD compares calculated CMAC and received CMAC	= 465D6D4737EBE157? 465D6D4737EBE157

6.4.2 Application Keys

All available application keys are shown in [Table 49]. Please note that except KeyID.AppDAMDefaultKey, KeyID.AppTransactionMACKey, KeyID.VCProximityKey, KeyID.VCSelectMACKey and KeyID.VCSelectENCKey all keys are keys which are located in the AKS.

Table 49. Application Keys

Key Name	Nr	Key Type	Usage	To change
<u>KeyID.AppMasterKey</u>	00	KeyType.AES (AKS Key Type)	<ul style="list-style-type: none"> Changing application level keys (if required by KeySet1) Changing the KeySet1 (if allowed by KeySet1) Formatting an application (for delegated application only) Creating and deleting files (if required by KeySet1) Executing the <u>Cmd.GetFileIDs</u>, <u>Cmd.GetFileSettings</u> (if required by KeySet1) 	<ul style="list-style-type: none"> Use <u>KeyID.ChangeKey</u> or <u>KeyID.ChangeKey EV2</u> command Only after authentication with <u>KeyID.AppMasterKey</u> Can be made unchangeable by Key Set1
<u>KeyID.AppChangeKey</u>	**	KeyType.AES (AKS Key Type)	<ul style="list-style-type: none"> To change any key other than <u>KeyID.AppMasterKey</u> (of AKS) 	<ul style="list-style-type: none"> Use <u>KeyID.ChangeKey</u> or <u>KeyID.ChangeKey EV2</u> command Only after authentication with <u>KeyID.AppMasterKey</u>
<u>KeyID.AppRollKey</u>	**	KeyType.AES (AKS Key Type)	<ul style="list-style-type: none"> To roll to another application key set 	<ul style="list-style-type: none"> Only after authentication with <u>KeyID.AppMasterKey</u>
AppKeys	***	KeyType.AES (AKS Key Type)	<ul style="list-style-type: none"> To read, write and change file access rights 	<ul style="list-style-type: none"> Use <u>KeyID.ChangeKey</u> or <u>KeyID.ChangeKey EV2</u> command
<u>KeyID.AppCommitReaderIDKey</u>	**	KeyType.AES (AKS Key Type)	<ul style="list-style-type: none"> To commit the Reader ID Authentication with <u>KeyID.AppCommitReaderIDKey</u> is necessary before committing Reader ID 	<ul style="list-style-type: none"> Only after authentication with <u>KeyID.AppChangeKey</u>
<u>KeyID.AppDAMDefaultKey</u>	****		<ul style="list-style-type: none"> Holds the key initialization value and version of each key (of a key set) used in delegated application creation Initialize keys (value and version) of key set (of delegated application) also later with <u>Cmd.InitializeKeySet</u> 	<ul style="list-style-type: none"> Can be changed via <u>Cmd.SetConfiguration</u> (for the selected delegated application) Only after authentication with <u>KeyID.AppMasterKey</u>
<u>KeyID.AppTransactionMACKey</u>	****		<ul style="list-style-type: none"> To calculate the CMAC on data manipulation commands (for TMAC file operations) 	<ul style="list-style-type: none"> This key can only be changed by deleting and re-creating the TMAC file
<u>KeyID.VCProximityKey</u>	*****	KeyType.AES	<ul style="list-style-type: none"> Needed for the Proximity Check Feature (MAC calculation at the end of the protocol) 	<ul style="list-style-type: none"> Only after authentication with <u>KeyID.AppMasterKey</u>

** Any key number from the AKS or 0xE or 0xF

- for KeyID.AppCommitReaderIDKey, 0xE is RFU and 0xF means Cmd.CommitReaderID is disabled
- for KeyID.AppRollKey, E and F are not allowed)

*** All other key numbers of the AKS

**** Non-addressable

- KeyID.AppTransactionMACKey is provided at transaction mac file creation

***** The KeyID.VCKeys are not part of the key sets but directly addressable through the key numbers 0x21, 0x22 and 0x23. An application can override the KeyID.VCKeys from the PICC level.

7 Asymmetric Key Management

In the context of MIFARE DUOX, 2 variants of asymmetric keys are used:

- CARootKeys
- KeyPairs

CARootKeys are public keys only, which are the public keys of the certification authority that are used to validate the certificates during asymmetric authentication. Consequently, the card needs to have the public key that was used to sign the reader certificate and vice versa. Also, the CARootKey determines the access rights the authentication will grant if a certificate signed by this key is used. Each CARootKey has associated a set of access rights that can be further restricted by the certificate itself. This means, that a single CARootKey can be used with different certificates to grant different subsets of access rights.

KeyPairs are the actual ECC keys that are used for authentication. They can be either imported or directly generated in the MIFARE DUOX. In case they are generated, the public key part of the key pair is returned at creation. Its important to store this public key somewhere, as it is not stored inside the MIFARE DUOX. Once available, a certificate needs to be created, signed with the appropriate CA key and written into a file inside the MIFARE DUOX application for further use.

The common format for ECC keys used in context of MIFARE DUOX is raw byte format, where specifically the public key is represented in uncompressed byte notation (prefix 0x04).

7.1 ManageCARootKey

Cmd.ManageKeyPair is used to import a CARootKey into a MIFARE DUOX application or on PICC level. CARootKeys are used to sign and verify certificates, where on MIFARE DUOX itself only the public key is needed and present, as MIFARE DUOX does not create certificates, only verifies them during authentication. Therefore, a CARootKey in terms of MIFARE DUOX is always only the public part of a CARootKey pair.

Similar as for normal ECC keys, an application can hold at most 5 CARootKeys, on PICC level there can be 2.

Following parameters need to be specified when using Cmd.ManageKeyPair:

- KeyNo: number of the ECC key that is targeted. At PICC level, 2 key slots are available, at application level 5.
- CurveID: determines the used ECC curve, Nist P-256 or brainpoolP256r1
- AccessRights: This is a 2 byte long bit mask that defines which access rights will be granted after authentication with a key associated with a certificate signed with this targeted CARootKey. Contrary to symmetric keys, asymmetric keys can grant multiple access rights at once. The access rights specified with a CARootKey can be further restricted in the used certificate, details can be found in [Section 8](#)
- WriteAccess: Defined the CommMode that needs to be used and which access rights need to be granted previously to update a key entry.
- ReadAccess: Defined the CommMode that needs to be used and which access rights need to be granted previously to read the public key value with Cmd.ExportKey
- CLRFile: defines if certificate revocation is enabled and which file is used for the certificate revocation list
- CLRFileAID: in case the CRL file is stored in a different application on the MIFARE DUOX, the AID needs to be specified here.
- Issuer and IssuerLen: Optionally, a CARootKey can be associated with a trusted Issuer name. If set, this will be compared during authentication to the certificates Issuer field.

7.2 ManageKeyPair

The `Cmd.ManageKeyPair` is used for all operations that target ECC keys and key pairs inside a MIFARE DUOX application of on PICC level. The command can be used to either import an already existing key pair (i.e. the private key part of it) or generate a new key pair inside the MIFARE DUOX.

Following parameters need to be specified when using `Cmd.ManageKeyPair`:

- KeyNo: number of the ECC key that is targeted. At PICC level, 2 key slots are available, at application level 5.
- Option: determines if a key shall be generated, imported or updated.
- CurveID: determines the used ECC curve, Nist P-256 or brainpoolP256r1
- KeyPolicy: This 2 byte long bitmask determines what the targeted key can be used for. A good practise is always to limit the key usage to only what it really needs to have, to do not open up any misuse of the key.
- WriteAccess: Defined the CommMode that needs to be used and which access rights need to be granted previously to update a key entry.
- KUCLimit: A limit for key usage can be defined that limits the number of usage of the targeted key to the given value.

An example for the `Cmd.ManageKeyPair` can be found [Table 10](#) (import a key) and [Table 69](#) (generate key pair)

8 Certificate Management

MIFARE DUOX distinguishes between card certificates and reader certificates, which are outlined in the following sections.

8.1 Card Certificates

Card certificates are used to certify the public key of the keypair that is contained on the card. The certificates should be signed with the CA root public key that is present on the reader side for it to be able to verify the cards public key.

The MIFARE DUOX generally is agnostic of the format of card certificates, simply because the card does only store these certificates, they are only validated by the reader as part of the asymmetric authentication, i.e. the reader needs to know the certificate format and how to handle it.

Card certificates are stored on the MIFARE DUOX in `StdDataFiles` or `BackupDataFiles` in the same way as other data, they can be accessed with normal `Cmd.ReadData` and `Cmd.WriteData` commands, the file access rights apply as well. The first 3 byte in a file holding a certificate must contain the length of the certificate (CLEN) as number of bytes, LSB first. In case of the file size being greater than CLEN, all further data in the file is ignored during authentication.

MIFARE DUOX supports creating `StdDataFiles` and `BackupDataFiles` on application level as well as on PICC level, to also provide a certificate for PICC level ECC keys. The same rules as above also apply for PICC level card certificates.

8.2 Reader Certificates

Reader certificates are used during the asymmetric authentication to certify the readers public key. The reader certificate needs to be signed with the CA root key that is present on the card.

Reader certificates need to follow the X.509 v3 format [10], using ASN.1 distinguished encoding rules (DER). **Only mandatory fields** from [10] are supported, no optional fields can be used, otherwise MIFARE DUOX will not accept the certificate during authentication.

The exact definition of the reader certificate format can be found in the datasheet [1] in section 7.10.2.

Additionally, MIFARE DUOX supports restricting the access rights of a CA root key further per certificate used. This is done via an extension in the certificate. This is the **ONLY** extension that is allowed in the reader certificate, **NO OTHER extensions must be present**. The access rights extension is reflected by an OID in the NXP range: **1.3.6.1.4.1.28137.64.1**. Details about this extension are shown in [1] in section 7.6.4. It is **important to note** that the access rights granted by a certificate can only be a subset of the access conditions of the related CA root key. A certificate can NEVER grant an additional access right that is not present in the CA root key.

Also, a reader certificate can not be longer than **440 byte** for a single certificate, and **880 byte** for a certificate chain.

Below is an example reader certificate:

```
308201423081E9A003020102020401020304300A06082A8648CE3D0403023018
31163014060355042D030D00544553545F4341526F6F7430301E170D32343036
30373131333633355A170D3235313233313132333630365A301E311C301A0603
55042D031300547261696E696E675F546573745F436572743059301306072A86
48CE3D020106082A8648CE3D0301070342000425CDDFF5B699010090643F0E89
001E6DAC61FCD44DFFD6EF46D1F6AC7DFB7784438865C055B1B62EF3E96B11BD
65CA3080346D6E99F51C664F67CF10D1ED8C6DA31B30193017060A2B06010401
81DB6940010101FF040601112233FF3F300A06082A8648CE3D04030203480030
4502205A63883B12B162F377A27B86635A7F8E365E34AFF0AB1957BB5AC1242F
```

```
268396022100C05928C4B78512915ED495D75EC18D59888CC08C562AE6413CDD
AC06D55F8E13
```

This certificate can be visualized in any ASN.1 parser tool, e.g. <https://lapo.it/asn1js/>:

```
SEQUENCE (3 elem)
  SEQUENCE (8 elem)
    [0] (1 elem)
      INTEGER 2
      INTEGER 16909060
      SEQUENCE (1 elem)
        OBJECT IDENTIFIER 1.2.840.10045.4.3.2 ecdsaWithSHA256 (ANSI X9.62 ECDSA
algorithm with SHA256)
      SEQUENCE (1 elem)
        SET (1 elem)
          SEQUENCE (2 elem)
            OBJECT IDENTIFIER 2.5.4.45 uniqueIdentifier (X.520 DN component)
            BIT STRING (96 bit)
01010100010001010101001101010100010111110100001101000001010100100110111...
          SEQUENCE (2 elem)
            UTCTime 2024-06-07 11:36:35 UTC
            UTCTime 2025-12-31 12:36:06 UTC
          SEQUENCE (1 elem)
            SET (1 elem)
              SEQUENCE (2 elem)
                OBJECT IDENTIFIER 2.5.4.45 uniqueIdentifier (X.520 DN component)
                BIT STRING (144 bit)
0101010001110010011000010110100101101110011010010110111001100111010111...
              SEQUENCE (2 elem)
                SEQUENCE (2 elem)
                  OBJECT IDENTIFIER 1.2.840.10045.2.1 ecPublicKey (ANSI X9.62 public key
type)
                  OBJECT IDENTIFIER 1.2.840.10045.3.1.7 prime256v1 (ANSI X9.62 named
elliptic curve)
                  BIT STRING (520 bit)
0000010000100101110011011111111110101101101101001100100000001000000...
                [3] (1 elem)
                  SEQUENCE (1 elem)
                    SEQUENCE (3 elem)
                      OBJECT IDENTIFIER 1.3.6.1.4.1.28137.64.1
                      BOOLEAN true
                      OCTET STRING (6 byte) 011122330100
                    SEQUENCE (1 elem)
                      OBJECT IDENTIFIER 1.2.840.10045.4.3.2 ecdsaWithSHA256 (ANSI X9.62 ECDSA
algorithm with SHA256)
                      BIT STRING (568 bit)
0011000001000101000000100010000001011010011000111000100000111011000100...
                  SEQUENCE (2 elem)
                    INTEGER (255 bit)
4088401446201564571568374593311176541021735800207433716361470870722541...
                    INTEGER (256 bit)
8700159769008273339635472786022685881369464576409953568448175477786581...
```

Above certificate shows the mandatory fields and one access right extension.

8.3 Certificate Generation

As MIFARE DUOX is using very widely used certificate standards like X.509 and ASN.1, certificates can be generated with many common tools. However, due to conventions that have evolved in the field, some tools automatically add extensions and fields to certificates, that MIFARE DUOX is not able to handle (when used as card certificates). Those restrictions might be (non-taxative list):

- CSN (Certificate Serial Number) longer than 4 byte
- unsupported optional, but widely used fields, e.g. SubjectKeyIdentifier, AuthorityKeyIdentifier
- unsupported extensions
- etc.

A common tool to work with certificates and general cryptography is the well known **openssl**. This section will give an example on how to generate certificates that can be used with MIFARE DUOX.

8.3.1 OpenSSL for creating certificates

OpenSSL can be used to create MIFARE DUOX compatible certificates. The version used for the below examples is: **OpenSSL 3.3.2 3 Sep 2024**.

As a prerequisite to also be able to create the custom access right extension, following `x509_DUOX_OID.ext` file needs to be present in the working folder:

```
[ CA ]
# X.509 extensions for a CA
subjectKeyIdentifier = none
authorityKeyIdentifier = none
1.3.6.1.4.1.28137.64.1 = critical,DER:011234560200
```

Following steps are required to generate the relevant keys including optional custom access rights in the openssl commandline

```
openssl ecparam -name prime256v1 -genkey -noout -out host_root_priv_key.pem
openssl ec -in host_root_priv_key.pem -outform DER -out host_root_priv_key.der
openssl ec -in host_root_priv_key.pem -pubout -out host_root_pubkey.pem
```

Above lines will create the private and public key in PEM file format, as well as the private key in DER file format. Those file formats are very common and can be viewed with several tools. One example tool for viewing is the ASN.1 Editor, available here: <https://github.com/PKISolutions/Asn1Editor.WPF>

As a next step, a csr (Certificate Signing Request) is created using the following line:

```
openssl req -new -sha256 -key host_root_priv_key.pem -out host_root_sign_req.csr
-subj "/O=AnyCompany/CN=NXP RootCA"
```

The following line will now create the final certificate (in this case, its a self signed certificate, as the same private key as generated above is used for signing):

```
openssl x509 -extfile x509_DUOX_OID.ext -extensions CA -signkey
host_root_priv_key.pem -in host_root_sign_req.csr -req -days 3650 -set_serial
0x00000001 -out host_root_cert.pem
```

or, alternatively, .der format can be used:

```
openssl x509 -extfile x509_DUOX_OID.ext -extensions CA -signkey  
host_root_priv_key.pem -in host_root_sign_req.csr -req -days 3650 -set_serial  
0x00000001 -outform DER -out host_root_cert.der
```

To convert a .pem file into .der, following line can be used as well:

```
openssl x509 -in host_root_cert.pem -out host_root_cert.der -outform DER
```

The custom extension defining the access right restrictions is defined via the 2 parameters -extfile giving the file name of the x509_DUOX_OID.ext file and -extensions giving the identifier of the extension to add, here it is "CA".

The result of above lines is a host_root_cert.pem file that contains a self-signed certificate for the above created public key. The certificate again can be viewed in the above mentioned ASN.1 Editor, or simply in the command line with the following command:

```
openssl x509 -in host_root_cert.pem -text -noout
```

The result will look similar like this:

```
Certificate:  
  Data:  
    Version: 3 (0x2)  
    Serial Number: 1 (0x1)  
    Signature Algorithm: ecdsa-with-SHA256  
    Issuer: O=AnyCompany, CN=NXP RootCA  
    Validity  
      Not Before: Sep 25 11:01:24 2024 GMT  
      Not After : Sep 23 11:01:24 2034 GMT  
    Subject: O=AnyCompany, CN=NXP RootCA  
    Subject Public Key Info:  
      Public Key Algorithm: id-ecPublicKey  
      Public-Key: (256 bit)  
      pub:  
        04:a9:c3:e2:dd:51:dd:3f:ef:fd:d4:c9:84:dd:20:  
        6a:30:0c:f2:ee:66:53:f0:3b:7c:53:1f:0a:2e:bb:  
        cd:18:88:5d:08:23:a4:63:b4:e1:d1:a9:d4:86:04:  
        82:8a:32:65:9c:b8:95:7d:08:f3:91:ab:d5:be:21:  
        1e:07:c4:75:e9  
      ASN1 OID: prime256v1  
      NIST CURVE: P-256  
    X509v3 extensions:  
      1.3.6.1.4.1.28137.64.1: critical  
      ..4V..  
    Signature Algorithm: ecdsa-with-SHA256  
    Signature Value:  
      30:44:02:20:02:a0:be:7c:f8:15:d2:ef:0e:9e:0a:aa:ed:d5:  
      36:aa:e0:c3:86:c7:7b:06:d4:26:64:99:98:77:1a:88:b2:a4:  
      02:20:72:01:9b:2f:eb:66:a7:43:24:bd:15:79:fe:d6:2f:c4:  
      65:4d:db:6b:64:55:b2:ea:1f:0d:55:fb:2b:5c:25:e1
```

Unfortunately, openssl as such is not able to output the plain hex values that are needed for operation with MIFARE DUOX, however there are tools available that can do this conversion. Above mentioned ASN.1 Editor does show the certificate in hex formatting, or for Windows users, the Powershell command "Format-hex" can be used on a DER encoded certificate (.der file):

```
Format-Hex host_root_cert.der
```

will result in:

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00000000	30	82	01	5D	30	82	01	04	A0	03	02	01	02	02	01	01	0.]0... ..
00000010	30	0A	06	08	2A	86	48	CE	3D	04	03	02	30	2A	31	13	0...*Hî=...0*1.
00000020	30	11	06	03	55	04	0A	0C	0A	41	6E	79	43	6F	6D	70	0...U...AnyComp
00000030	61	6E	79	31	13	30	11	06	03	55	04	03	0C	0A	4E	58	any1.0...U...NX
00000040	50	20	52	6F	6F	74	43	41	30	1E	17	0D	32	34	30	39	P RootCA0...2409
00000050	32	35	31	31	30	31	32	34	5A	17	0D	33	34	30	39	32	251101242...34092
00000060	33	31	31	30	31	32	34	5A	30	2A	31	13	30	11	06	03	3110124Z0*1.0...
00000070	55	04	0A	0C	0A	41	6E	79	43	6F	6D	70	61	6E	79	31	U...AnyCompany1
00000080	13	30	11	06	03	55	04	03	0C	0A	4E	58	50	20	52	6F	.0...U...NXP Ro
00000090	6F	74	43	41	30	59	30	13	06	07	2A	86	48	CE	3D	02	otCA0Y0...*Hî=.
000000A0	01	06	08	2A	86	48	CE	3D	03	01	07	03	42	00	04	A9	...*Hî=....B..©
000000B0	C3	E2	DD	51	DD	3F	EF	ED	D4	C9	84	DD	20	6A	30	0C	ÅÅYQY?iYÔÉY j0.
000000C0	F2	EE	66	53	FC	3B	7C	53	1F	0A	2E	BB	CD	18	88	5D	øifSø; S...»Í.]
000000D0	08	23	A4	63	B4	E1	D1	A9	D4	86	04	82	8A	32	65	9C	.#*c'áÑ©Ô.2e
000000E0	B8	95	7D	08	F3	91	AB	D5	BE	21	1E	07	C4	75	E9	A3).ó«Ô¾!...Äuéf
000000F0	1B	30	19	30	17	06	0A	2B	06	01	04	01	81	DB	69	40	.0.0...+....Ôie
00000100	01	01	01	FF	04	06	01	12	34	56	02	00	30	0A	06	084V..0...
00000110	2A	86	48	CE	3D	04	03	02	03	47	00	30	44	02	20	02	*Hî=....G.0D. .
00000120	A0	BE	7C	F8	15	D2	EF	0E	9E	0A	AA	ED	D5	36	AA	E0	¾ ø.Òi...ªiÔ6ªà
00000130	C3	86	C7	7B	06	D4	26	64	99	98	77	1A	88	B2	A4	02	ÃÇ{.Ô&dw.²ª.
00000140	20	72	01	9B	2F	EB	66	A7	43	24	BD	15	79	FE	D6	2F	r./éfSC\$½.yþÖ/
00000150	C4	65	4D	DB	6B	64	55	B2	EA	1F	0D	55	FB	2B	5C	25	ÅeMÛkdU²ê..Uû+\%
00000160	E1																á

In order to display the key value of a previously generated key, the following command might be useful

```
openssl pkey -inform der -in host_root_priv_key.der -noout -text
```

8.4 Certificate Revocation

MIFARE DUOX supports a mechanism to revoke specific reader certificates by the use of a CRL (certificate revocation list). A CRL is linked to a CARootKey and can either revoke all CSNs (certificate serial numbers) below a given number, single CSNs from a list or both together. Additionally, a CRL always has a version assigned, which can be used to avoid replaying older file versions. The version is written at each update of the CRL, and can be configured to either be strictly monotonically increasing or even increasing by one each time, to avoid missing any update. A signature calculated with a CARottKey can be added to the write command, which means that no single reader has control over the CRL file content, only the CA has.

A CRL file is created as a normal StdDataFile or BackupDataFile, but with a flag that identifies it as CRL file. The further configuration is then in a next step done with Cmd.ChangeFileSettings. The CSNs are written using a Cmd.WriteData. The exact definition of the CRL file can be reviewed in the datasheet [1]

[Table 50](#) shows the creation of a CRL and liked CARootKey (CommMode is encrypted, but plain data is shown in comments):

Table 50. CRL File creation

Step	Action	Direction	Command	Comment
1	CreateStdDataFile	>	CD01A0001000E000030017623E30E26BF631	Create a StdDataFile with FileNumber = 0x01 and FileOption = 0x10, indicating its a CRL file. The File will be freely readable, Comm Mode plain.
2	Success + CMAC	<	00B2966CCEC222375B	
3	ChangeFileSettings	>	5F011EDDB23B07EE50971576D6757EC479BB81FB6E2537027A83	Changes the file properties to enable the CRL file. The command is using CommMode.Full, but essentially this command is not changing any of the file properties, it just make sure that the CRLOption and CSNLength is set properly (0x04) If a CRLSignature is required, it needs to be enabled in the CRLOptions and the number of the CARootKey needs to be provided as well.
4	Success + CMAC	<	00D39B2D6A025050C9	
5	WriteData	>	3D0101000400000400000000000001	This WriteData command writes a certificate serial number (CSN) into the file. This 4 byte long number is 0 0000001 in this example. It is written with an offset of 040000 (4 byte, LSB first), meaning its sparing out the 4 byte of the file, which is reserved for holding a CSN "limit" number, that would disable all CSNs below this one.
6	Success	<	00	

Table 50. CRL File creation...continued

Step	Action	Direction	Command	Comment
7	ManageCARootKey	>	48000C01003E3E 81123456 EF0E8B3CB29D8 D69421840C7D9DDA5661AC66362453FA09AC18533 D0F0E77205E5202EAEF4BA1A468C46F88318858 BBF65ED00B896C755C787C121F88A9DB9FC2 AA19327CD6B065F849BDF9E561937BEFC2955B1 ABD6C6	The ManageCARoot Key command links the just create CRL File to this CARootKey. The bold parameter marked on the left indicates that CRL is enabled (8xxxxxx), the CRLFile number as created before (x1xxxxx) and the AID of the application in which the CRL File is located (xx123456)
8	Success + CMAC	<	007B2BFF0EAB764746	

9 Application Management

MIFARE DUOX supports two types of applications: Conventional applications and delegated applications. A new NXP chip which comes from factory, has no application in it. Applications can be created as long as user memory is available (there is no limit on number of applications that can be created).

9.1 Conventional Application Management

Conventional applications are usually managed by the owner of the PICC. They are created at PICC personalization. Conventional applications are created using `Cmd.CreateApplication`. The communication mode of `Cmd.CreateApplication` is `CommMode.MAC`.

Parameters of `Cmd.CreateApplication` are described in Ref. [1]. Please note that besides `KeySet1`, other parameters are fixed (cannot be changed once the application was created). `KeySet1` can be changed later using `Cmd.ChangeKeySettings` (if bit 3 is not set to zero). If the application is created with multiple key sets and different key sets are personalized to different key types, `KeySet2` (bits 7 - 6) gets updated automatically if key set rolling to a new key set takes place. When creating conventional applications, there is no way to restrict the memory usage, i.e. files can be created consuming the whole user memory (available) of the PICC.

A PICC can be configured in a way (bit 2 of `PICCKeySettings` is set to 0) that creation of conventional application needs prior authentication with `KeyID.PICCMasterKey`. The default configuration (bit 2 of `PICCKeySettings` is set to 1) allows anyone to create conventional applications without prior authenticating with `KeyID.PICCMasterKey`.

9.1.1 Example: Create Conventional Application in EV2 Secure Messaging

`Cmd = CA`

`AID = 1FC1C2`

`KeySet1 = 1F` (Change key: 0x01, all configuration enabled)

`KeySet2 = B5` (AES key type, ISO 7816 IDs are used for app and files, Key set param. present, 5 keys per key set)

`KeySet3 = 01` (App specific capability data disabled, App specific VC keys disabled, Application key sets enabled)

`AKSVersion = 00`

`NoKeySets = 04`

`MaxKeySize = 10` (16 bytes)

`AppKeySetSett = 02`

`ISOFileID = 1122`

`ISODFName = 010203040506`

`KeyID.SesAuthMACKey = 936F81529886567E164402DD6531F5E8`

`Current CmdCtr = 0200` (LSB first)

`TI = 0EF80413`

`IV = 00000000000000000000000000000000`

Table 51. Create Conventional Application in EV2 Securing Messaging

Step	Command		Data Message
1	Calculate CMAC on CA0200 0EF804131FC1C21FB501000410021122010203040506 (Cmd CmdCtr TI AID KeySet1 KeySet2 KeySet3 AKSVersion NoKeySets MaxKeySize AppKeySetSet ISOVID ISODFName)	=	68CB54266E0ED482
2	<u>Cmd.CreateApplication</u> C-APDU	>	CA1FC1C21FB50100041002112201020304050 668CB54266E0ED482
3	Updated CmdCtr	=	0300
4	R-APDU	<	00787B2EFA996DCA91
5	Calculate CMAC over 003000EF80413 (RC CmdCtr TI)	=	787B2EFA996DCA91
6	Compare calculated CMAC and received CMAC from the PICC	=	787B2EFA996DCA91 ? 787B2EFA996DCA91

9.1.2 Delete Conventional Application

A conventional application can be deleted permanently using Cmd.DeleteApplication. If bit 2 of PICCKeySettings is set to 0, the deletion of conventional applications always needs prior authentication with KeyID.PICCMasterKey. If bit 2 of PICCKeySettings is set to 1 (default settings), the conventional application can be deleted after active authentication with KeyID.PICCMasterKey or KeyID.AppMasterKey. The command Cmd.DeleteApplication requires CommMode.MAC.

After deletion of a conventional application, only the memory blocks used as application overhead (1 block of 32 bytes) are released and available to be reused for application creation. Memory blocks used for files and key storage are not released and cannot be reused. The complete memory can be freed and released only by using Cmd.Format at PICC level, which deletes all applications, releases the complete memory and makes it possible to reuse.

Note: Single conventional applications cannot be formatted.

9.2 Delegated Application Management (DAM) - using symmetric keys

In conventional application management, if a PICC owner wants to sell application space to a third party, there are a few options:

1. The PICC owner can issue the card empty and applications can be installed post-issuance (bit 2 of PICCKeySettings is set to 1 – Cmd.CreateApplication doesn't need any prior authentication with KeyID.PICCMasterKey, but in this case anybody can create applications on the PICC).
2. The PICC owner can share the KeyID.PICCMasterKey with the third party. The consequence is, that the third party can do anything on the PICC.
3. The PICC owner can pre-personalize the card, i.e. create an application for the third party according to their needs. But after sharing the application master key, the PICC owner will not have any control over the memory size.

With the Delegated Application Management (DAM) feature of MIFARE DUOX, the PICC owner can delegate application creation to third parties (application providers), avoiding the above mentioned restrictions. Delegated applications are limited by a maximum quota (maximum memory consumption), so that the card issuer can manage the memory space amongst different delegated application providers.

9.2.1 Steps needed for enabling Delegated Application Management

In order to make the delegated application management possible, there are mainly three steps required:

1. Personalization of the card by the card issuer
2. Agreement between card issuer and application provider regarding application creation and quota limit
3. Application creation by application provider

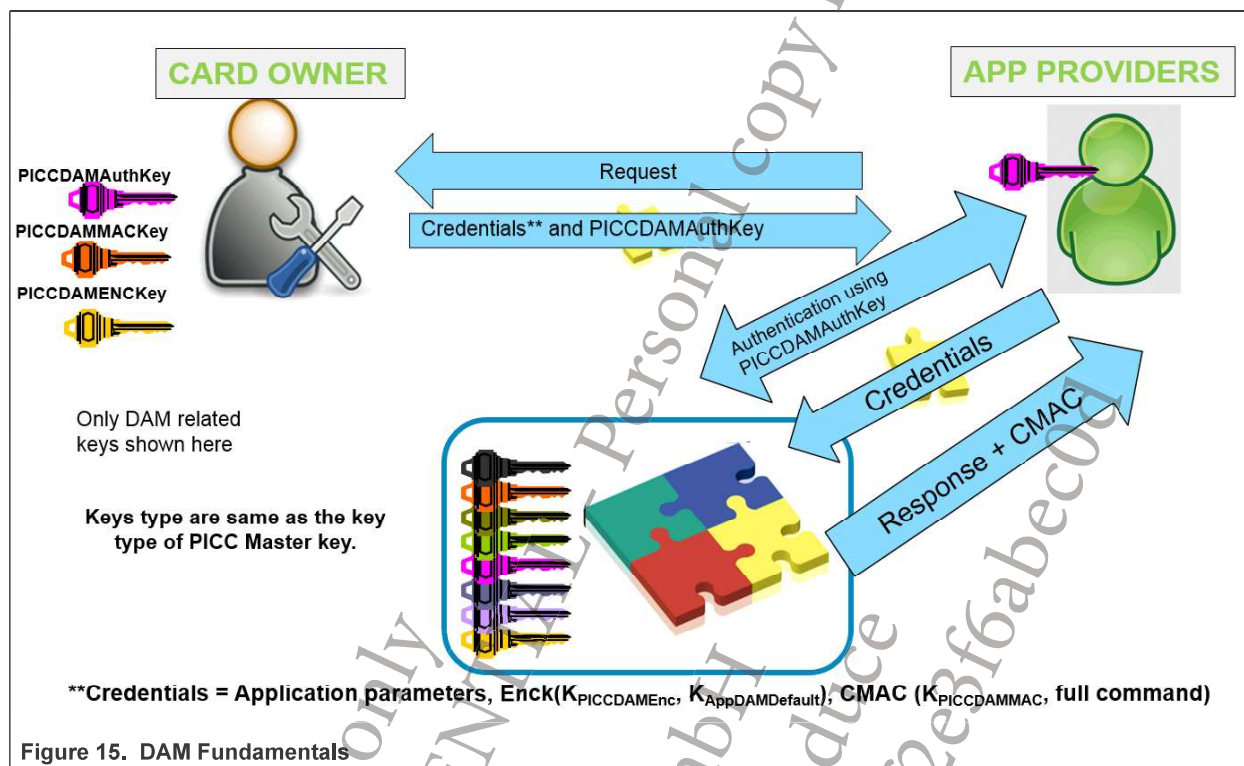
9.2.2 Responsibilities of the card issuer

It's the responsibility of the card issuer to prepare the card for DAM through personalizing the card with the therefore required keys.

1. PICC owner first prepares the PICC for delegated application creation by enabling the DAM keys (see [Section 9.2.4.2](#)).
2. PICC owner generates a [Section 9.2.5.1](#) (for the application to be created on the PICC) based on all parameter which were agreed with the application provider (including encrypted KeyID.AppDAMDefaultKey and its version).
3. PICC owner then shares KeyID.PICCDAMAuthKey, the encrypted KeyID.AppDAMDefaultKey, and the [Section 9.2.5.1](#) with the application provider who can later create its application.

9.2.3 DAM Fundamentals

Instead of sharing KeyID.PICCMasterKey or keeping the PICC open, a PICC owner shares KeyID.PICCDAMAuthKey, the encrypted KeyID.AppDAMDefaultKey, and a cryptogram called DAMMAC (CMAC calculated over all agreed parameters including the encrypted KeyID.AppDAMDefaultKey) with the application provider (see DAMMAC generation in [Section 9.2.5.1](#)). An application provider can then create its own application on a PICC during normal personalization or even on a PICC which is already deployed in the field. DAM fundamentals and the exchanged data, is illustrated in [\[Figure 15\]](#) and also described in the following sections.



When using Delegated Application Management, the PICC owner can always delete the applications after authentication with `KeyID.PICCMasterKey`. However, it cannot do anything with the application or read any data from the application as long as he does not know the application keys (after the application provider has changed them). The PICC owner shall maintain a database of AIDs, quota limit, slot number, version etc.

Each delegated application can have its own `KeyID.AppDAMDefaultKey`, which ensures isolation between the different application providers at all time.

9.2.4 DAM Step 1: Personalization of the Card by the Card Issuer

For the personalization of the PICC, the card issuer needs to generate the three DAM keys which are described in [Section 9.2.4.1](#). The steps for enabling them are listed in [Section 9.2.4.2](#).

9.2.4.1 DAM Keys

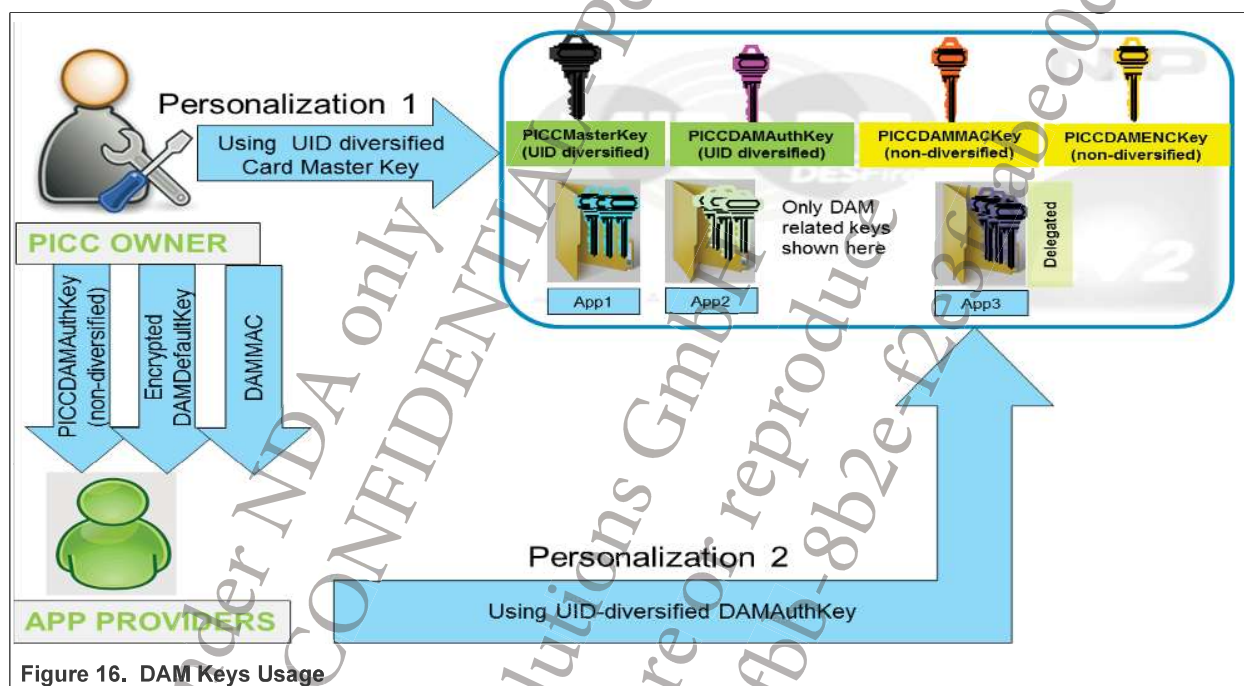
`KeyID.PICCDMAAuthKey`, `KeyID.PICCDAMENCKey` and `KeyID.PICCDAMMACKey` are called DAM keys.

- `KeyID.PICCDMAAuthKey` will be shared with the application provider. The application provider needs to authenticate with this key to be able to create a delegated application. This key can be stored diversified by UID on the PICC. In that case, the non-diversified key is shared.
- `KeyID.PICCDAMMACKey` is used to generate the DAMMAC. This key is only known by the card issuer. Diversifying this key can be useful in order to control the amount of cards which are accessed by the service provider.
- `KeyID.PICCDAMENCKey` (usually not diversified) is used to encrypt the `KeyID.AppDAMDefaultKey`. This key is only known by the card issuer.

Note: In general `KeyID.PICCDAMMACKey` as well as `KeyID.PICCDAMENCKey` will not be diversified by the UID, as in this case the card issuer would need to provide the application provider with a different cryptogram for each card. For this reason generally the `KeyID.AppDAMDefaultKey` will also not be diversified.

In case the application provider also wants to use the Proximity Check feature, the card issuer also needs to personalize the card with the keys for Virtual Card Selection and Proximity Checking, and share them with the application provider.

[Figure 16] shows the DAM key usage in two possible scenarios. In scenario 1 (Personalization 1), the PICC owner is creating its own applications using `KeyID.PICCMasterKey` diversified by UID, and in scenario 2 (Personalization 2), the application provider is creating its application using `KeyID.PICCDAMAuthKey` diversified by UID. Please note that the PICC owner can also introduce a DAM scheme with charge-per-use by using the diversified `KeyID.PICCDAMMACKey` and/or `KeyID.PICCDAMENCKey`. In that case he will need to deliver a cryptogram for each card.



9.2.4.2 Enabling the DAM keys

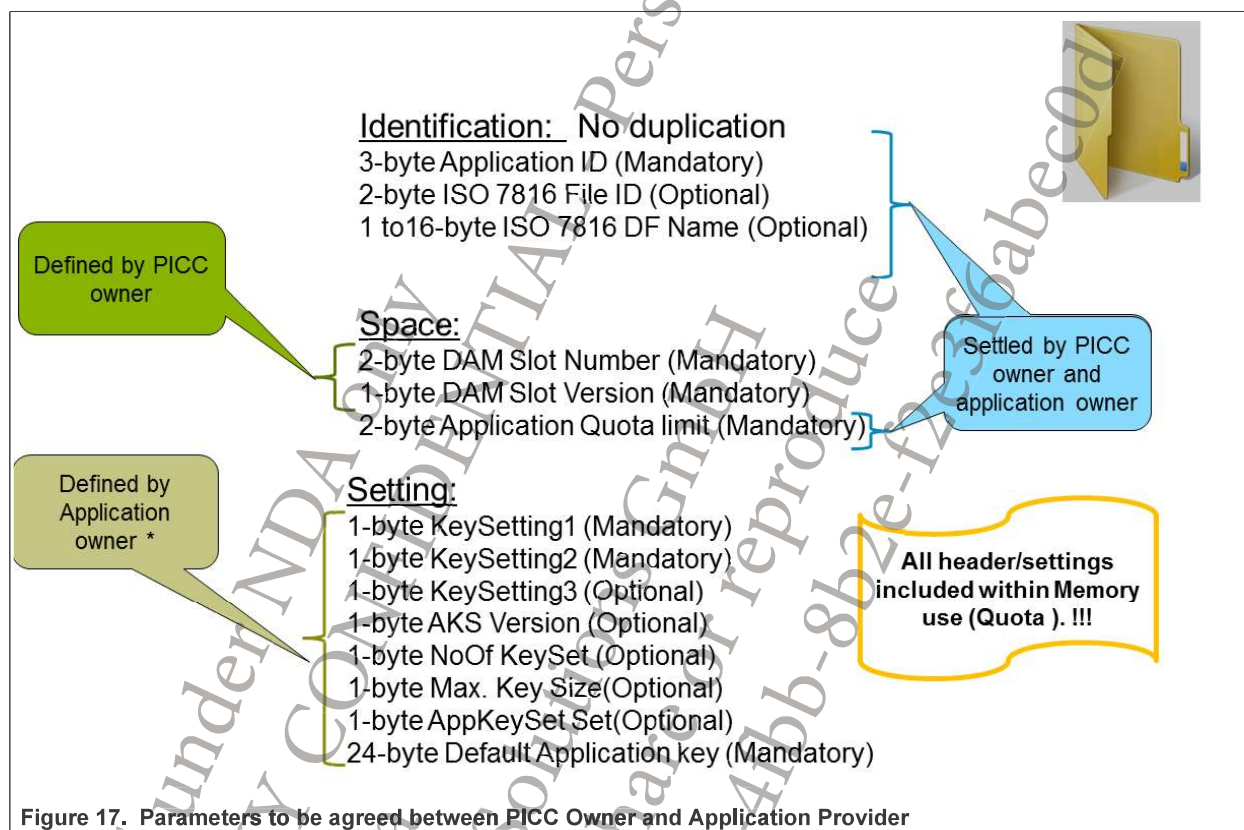
The DAM keys are of the same type as `KeyID.PICCMasterKey`. They are disabled by default and have to be enabled by changing them using `Cmd.ChangeKey` or `Cmd.ChangeKeyEV2`. If the PICC master key is changed to another type, the DAM Keys gets disabled (reset to zeros) and need to be enabled again.

Note: If only the PICC Master Key value gets changed and the key type remains the same, the DAM keys also remain as they are.

9.2.5 DAM Step 2: Agreement between the PICC Owner and Application Provider (On the DAM Parameters)

The PICC owner and the application provider need to agree on application parameters before the PICC owner can sell memory space to the application provider.

The parameters which need to be agreed upon include: AID, QuotaLimit (maximum of consumable memory for the application provider), KeySet1, KeySet2, and eventually more. Basically all parameters which are necessary for the application creation need to be agreed upon. The parameters which need to be agreed upon are shown in [Figure 17]. Please also look into Ref. [1] for the detailed description of all parameters. Some DAM related parameters are described in the following sections.



The card issuer needs to supply the following items to the application provider:

1. The `KeyID.PICCDAMAuthKey` so that the application provider can authenticate to the card.
2. The application parameters together with the DAMMAC over these parameters, which is generated using the `KeyID.PICCDAMMACKey` (DAMMAC generation is described in [Section 9.2.5.1](#)).
3. The required Virtual Card Selection keys and Proximity Check keys if needed.
4. The `KeyID.AppDAMDefaultKey` which is needed to initialize the application keys of the delegated application (this key will be delivered encrypted with the `KeyID.PICCDAMENCKey` as described in [Section 9.2.5.2](#)).

9.2.5.1 DAMMAC Generation

As mentioned earlier, an 8-byte CMAC is calculated over all application parameters including the encrypted `KeyID.AppDAMDefaultKey`.

CMAC calculation is done according to EV2 Secure Messaging MAC Calculation with the IV being a zero byte IV.

Details of the DAMMAC calculation are available in Ref. [\[1\]](#) Section 6.6.2.2.2.

9.2.5.2 Encryption of the KeyID.AppDAMDefaultkey

The `KeyID.AppDAMDefaultKey` is encrypted by the card issuer using the `KeyID.PICCDAMENCKey` and provided to the application provider. The application provider needs to use it as parameter `Enck` in `Cmd.CreateDelegatedApplicationPart2`.

The key value concatenated with the key version of the `KeyID.AppDAMDefaultKey` is padded with random numbers in the front in order to make it 32 byte aligned. These 32 byte are then encrypted using the `KeyID.PICCDAMENCKey` using a zero bytes IV. An example for this is shown in [Section 9.2.8](#).

9.2.6 DAM Step 3: Application Provider creates its Application

In order to create an application on the PICC, the application provider needs to power the PICC and eventually do Virtual Card Selection and Proximity Check (only if needed).

After this, he needs to authenticate against the PICC using the `KeyID.PICCDAMAAuthKey` and then can create the application using the two parts of the command `Cmd.CreateDelegatedApplication` together with the agreed parameters as well as the DAMMAC. The PICC verifies the received DAMMAC using the `KeyID.PICCDAMMACKey`. Only if this validation succeeds, the application can be created, otherwise the command is rejected. The detailed functioning of the command is described in Ref. [\[1\]](#) in [Section 6.6.2.2](#).

After the creation of the application and the needed files, the application provider will change the application keys and settings as required. This will give him full control over the application (PICC owner cannot do anything with the application, except deleting the application for example if the contract with the application provider expires).

9.2.7 Important DAM Parameters

In this section some important parameters which are needed for the delegated application management are described briefly. They are also discussed in Ref. [\[1\]](#).

9.2.7.1 DAM Slot Number

A delegated application is associated to a DAM slot number. It allows the PICC owner to target a specific slot of the application memory space. The PICC is initially not divided into slots. When a delegated application is created, it is created in the lower part of the available memory location. The next delegated application having a different slot number will be created in the next available memory location. However, the slot numbers don't have to be sequential.

A DAM slot number is always also associated with the requested quota limit (maximum size that can be used by the application). As after application deletion it is possible to reuse slots, this fact is very important. Only if the slot number and the quota limit are the same for the new application to be created (as it was for the old application), a new application can be created in the free slot.

It's the responsibility of the PICC owner to maintain the slot number and the quota limit associated to it.

9.2.7.2 DAM Slot Version

After successful delegated application creation, each slot with a DAM slot number is also associated with a DAM slot version. In case a delegated application becomes obsolete, another delegated application can be created in the slot with the same DAM slot number but only using a strictly higher DAM slot version. The slot version prevents the re-creation of obsolete delegated applications with a lower DAM slot version targeting the same DAM slot number.

This mechanism makes it impossible for the application provider to install the same delegated application using the same DAM slot version again. It allows the PICC owner to delete a delegated application without the need to reuse the free slot immediately for creating a new delegated application in the same slot. For a re-installment of the old application, the application needs a new cryptogram including the new, higher DAM slot version.

The only exception therefore is the DAM slot version set to 0xFF. This means, that the targeted slot can be used over and over again for the installation of delegated applications (as long as the application fulfills other agreed obligations). This allows PICC owners to sell the same memory slot to a number of different application providers (who offers short lived applications, for example event tickets like cinema tickets, single trip transporting tickets, or any other type of one-time validity tickets). The end user, e.g. the card holder and daily user, can decide which application he wants to install in the specific slot and can also delete and re-install them according to his needs.

9.2.7.3 Quota Limit

The quota limit is the assigned/reserved memory for a specific slot. It indicates the maximum memory the application can consume in number of 32 byte blocks.

The quota limit is calculated considering all application parameters including all the overhead (application administration and key storage). It is provided in number of 32 bytes blocks.

Example: For a 1024 byte sized delegated application, 32 memory blocks ($32 * 32 \text{ byte} = 1024 \text{ byte}$) will be necessary and the resultant quota limit will be "0x20 0x00" (LSB first).

Once a delegated application shall be created:

1. The assigned quota limit is reserved from the complete memory of the PICC.
2. The overhead (for this dedicated application) is reduced from the requested quota limit and the remainder of the memory blocks is available for the file creation.

9.2.7.4 Application DAM Default Key

The default key of a delegated application (KeyID.AppDAMDefaultKey) has no type associated to it and always consists of a 32 byte string for its value and a 1 byte key version. The 32 byte value KAppDAMDefault represents the encoded key (independent of the targeted key type) and the version encodes the KeyVerAppDAMDefault.

In case of AES128, only the 16 leftmost bytes represent the actual key.

This key (32 bytes) and key version can be also chosen by the application provider and provided to the PICC owner.

9.2.8 Example: Create Delegated Application

The command is sent in two frames using the MIFARE DUOX native chaining. In this example the focus is put on the generation of the EncK as well as the DAMMAC.

Prepare command part 1:

CMD = C9

AID = 1FD1D2

DAMSlotNo = 0200, (LSB first)

DAMSlotVersion = 01

QuotaLimit = 2000 (LSB First, decimal 1024 = 32 blocks each 32 bytes)

KeySet1 = 1F (Change key = 0x01, all configuration enabled)

KeySet2 = B5 (AES key type, ISO 7816 IDs are used for app and files, Key set param. present, 5 keys per key set)

KeySet3 = 01 (App specific capability data disabled, App specific VC keys disabled, Application key sets enabled)

AKSVersion = 00

NoKeySets = 04

MaxKeySize = 10 (16 bytes)

AppKeySet = 02

ISOFileID = 1122

ISODFName = 010203040506

Resulting command part 1: C91FD1D202000120001FB501000410021122010203040506

Prepare command part 2:

1. Encryption of AppDAMDefaultKey

PICCDAMENCKey = FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

AppDAMDefaultKey (24 byte) & Version (1 byte) = 00112233445566778899AABBCCDDEEFF001122334455667700

Random number (7 Byte) = FF710170C69EF0

IV = 00000000000000000000000000000000 (always)

2. DAMMAC Generation

IV = 00000000000000000000000000000000 (always)

PICCDAMMACKey = FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF (KeyType.AES → therefore generation according to EV2 Secure Messaging)

Table 52. Encryption of KeyID.AppDAMDefaultKey

Step	Command		Data Message
1	AppDAMDefaultKey	=	00112233445566778899AABBCCDDEEFF001122334455667700
2	Random Number Rnd	=	FF710170C69EF0

Table 52. Encryption of KeyID.AppDAMDefaultKey...continued

Step	Command		Data Message
3	Rnd AppDAMDefaultKey	=	FF710170C69EF000112233445566778899AABBCCDD EEFF001122334455667700
4	IV	=	00000000000000000000000000000000
5	PICCDAMEncKey	=	FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
6	EncK = E(K _{PICCDAMEncKey} , Rnd AppDAMDefaultKey)	=	FFDAB2B5080B7239DFA6D782607E7B0458297C8 EDEC21509D8B3D31240664E8D

Table 53. DAMMAC Generation

Step	Command		Data Message
1	Input Data to calculate DAMMAC (Cmd AID DAMSlotNo DAMSlotVersion QuotaLimit KeySet1 KeySet2 KeySet3 AKSVersion NoKeySets MaxKeySize AppKeySetSett ISOFileID ISODFName EncK)	=	C91FD1D202000120001FB501000410021122010203 040506FFDAB2B5080B7239DFA6D782607E7B0458297C8 EDEC21509D8B3D31240664E8D
2	DAMMAC = MAC _{DAM} (K _{PICCDAMMACKey} , Input Data)	=	0D62E6F221C981C7

Resulting command part 2:

FFDAB2B5080B7239DFA6D782607E7B0458297C8EDEC21509D8B3D31240664E8D0D62E6F221C981C7

9.2.8.1 Create Delegated Application Example (in EV2 SM)

KeyID.SesAuthMACKey (from Authentication with KeyID.PICCDAMAuthKey) =
446B1DC31F8C136605BB9A527F4B23B9

Current CmdCtr = 0000 (LSB first, reset to 0000)

TI = 26973400

Table 54. Create Delegated Application using EV2 Secure Messaging

Step	Command		Data Message
1	Data to calculate CMAC (Cmd CmdCtr TI command part1 command part2)	=	C90000269734001FD1D202000120001 FB501000410021122010203040506FFDAB2 B5080B7239DFA6D782607E7B0458297C8 EDEC21509D8B3D31240664E8D0D62E6F221 C981C7
2	CMAC from last step	=	8AE8B0874A42EC6D
3	Cmd.CreateDelegatedApplication C-APDU (part 1)	>	C91FD1D202000120001 FB501000410021122010203040506
4	R-APDU	<	AF
5	Cmd.CreateDelegatedApplication C-APDU (part 2 => Encrypted DAMDefaultKey)	>	AFFFDAB2B5080B7239DFA6D782607E7 B0458297C8EDEC21509D8B3D31240664E8D0 D62E6F221C981C78AE8B0874A42EC6D
6	Updated CmdCtr	=	0100

Table 54. Create Delegated Application using EV2 Secure Messaging...continued

Step	Command		Data Message
7	R-APDU (status CMAC)	<	0019802F4A92D58F81
8	Data for calculating CMAC on response (RC CmdCtr TI)	=	00010026973400
9	CMAC to be received	=	19802F4A92D58F81
10	PCD compares calculated CMAC and received CMAC	=	19802F4A92D58F81 ? 19802F4A92D58F81

The CmdCtr from Step 6 gets updated after validating the command (before sending the response). Updated CmdCtr will be used in next calculation

9.2.9 Delete Delegated Application

Delegated applications can be deleted permanently using `Cmd.DeleteApplication`. If b2 of `PICCKeySettings` is set to 0, deletion of delegated application needs prior authentication with `KeyID.PICCMasterKey`. If b2 of `PICCKeySettings` is set to 1 (default settings), delegated application can be deleted with an active authentication with `KeyID.PICCMasterKey` or `KeyID.AppMasterKey`.

After deletion of a delegated application, memory blocks used for this delegated application are released and can be reused, but only for another delegated application with exactly same `QuotaLimit`, in the same `DAMSlotNo` with a higher `DAMSlotVersion`.

9.2.10 Format Delegated Application

If a deleted application is formatted, the complete memory (that was previously used for files and data) is again available for that application to reuse. Formatting is done using `Cmd.Format` by the application provider after authentication with `KeyID.AppMasterKey`. The application keys and application configuration are not affected, only the file and data content is deleted.

9.3 Shared Application

In a multi-application environment, application providers (each having their own application) sometimes need to share common files (e.g.: a value file for an e-purse) or want to share files in order to maximize the efficiency of their applications. Data manipulation to these common files should be done in a single transaction so that rollback is consistent in case a tearing happens. Otherwise, it will be the case that value is debited but record (log) is not updated. The combination of all commands targeting two applications makes the transaction consistent and also faster, as only one `Cmd.CommitTransaction` is needed at the end of all file manipulations.

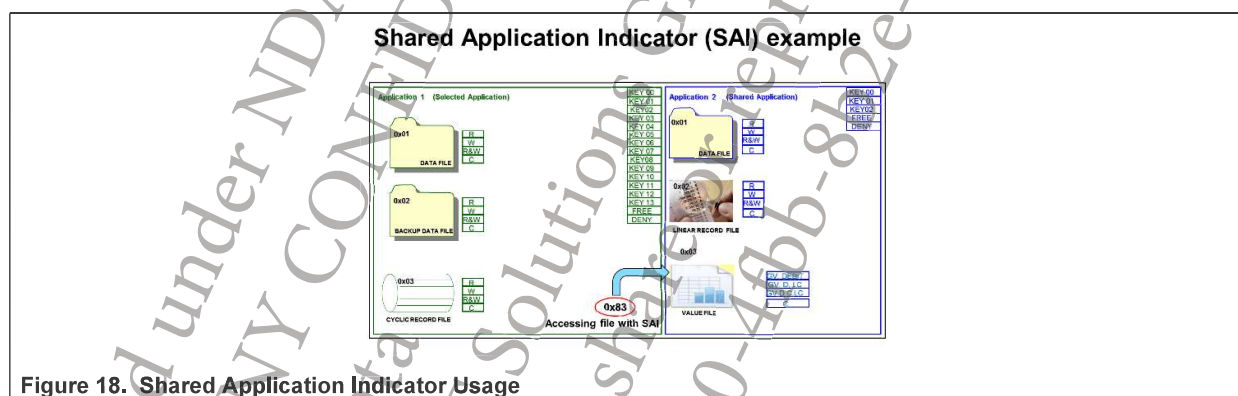
With the shared application feature of MIFARE DUOX, two independent applications can be selected using one `Cmd.SelectApplication`.

`Cmd.SelectApplication` accepts two application IDs as parameters' (primary application and secondary application). After selecting two applications, operations on keys or on files can be done on both applications. Only one of these two applications can have a TransactionMAC file inside the application.

After selection, until a commit transaction happens, all operations which are executed are considered as one transaction – it does not matter which of the two selected applications is targeted.

The FileNo, KeySetNo and KeyNo of the second application is accessed by using the so called Second Application Identifier (SAI). The SAI has the value 0x80 and when accessing any object in the second application, the number that want to be accessed needs to be XORed with the SAI. An example of this can be seen in [Figure 18](#). In this example, the file number that wants to be accessed in the secondary application originally has the FileID 0x03, but when using the SAI, the ID becomes 0x83.

The access rights of the files of the second application are according to the access rights of the second application. A commit transaction finalizes all the data manipulations in both applications. Please note that some commands (e.g. `Cmd.GetKeySettings`, full overview please see in Ref. [1]) always refer to the primary application.



10 File Management

MIFARE DUOX maintains user data in form of files of different types. The six different available types are:

- FileType.StandardData
- FileType.BackupData
- FileType.LinearRecord
- FileType.CyclicRecord
- FileType.Value
- FileType.TransactionMAC (not intended to store user data, but needed for the TMAC feature)

A multiple set of commands is available in order to manage files and file content. Files can be created, deleted, updated and information to files can be written and read out. File access can be restricted with an access rights management.

Data can basically be stored in 3 different ways: as raw data (byte wise), as value, and in form of records.

All file types except FileType.StandardData implement a backup mechanism that will be elaborated in detail in [Section 12](#).

At file creation, a file number needs to be defined which shall be unique for the file inside a specific application. Also the basic set of access rights as well as the communication mode (that has to be used later on when accessing file data) need to be set. If desired, more than one mandatory set of access rights can be enabled during file creation as well.

10.1 File Access Rights Management

File data is accessed with three different access rights FileAR.Read, FileAR.Write and FileAR.ReadWrite. Each of these access rights is permitting the use of a subset of commands, see the detailed list in [Table 55](#). In addition an access right called FileAR.Change is specified per file, permitting Cmd.ChangeFileSettings to change the file access rights. An access right is granted if at least one condition associated to it is satisfied.

Table 55. Command List associated with Access Rights

Read	Write	ReadWrite	Change
Cmd.ReadData	Cmd.WriteData	Cmd.ReadData	Cmd.ChangeFileSettings
Cmd.GetValue	Cmd.GetValue	Cmd.WriteData	
Cmd.Debit	Cmd.LimitedCredit	Cmd.GetValue	
Cmd.ReadRecords	Cmd.WriteRecord	Cmd.Debit	
Cmd.ISOReadRecord	Cmd.ClearRecordFile	Cmd.Credit	
Cmd.ISOReadBinary	Cmd.ISOAppendRecord	Cmd.LimitedCredit	
	Cmd.ISOUpdateBinary	Cmd.ReadRecords	
		Cmd.WriteRecord	
		Cmd.UpdateRecord	
		Cmd.ClearRecordFile	
		Cmd.ISOReadRecord	
		Cmd.ISOAppendRecord	
		Cmd.ISOReadBinary	
		Cmd.ISOUpdateBinary	
		Cmd.CommitReaderID	

Each file is associated with 1 up to 8 access conditions sets. Each set is containing three access conditions: one for each of FileAR.Read, FileAR.Write and FileAR.ReadWrite. The first set is called the mandatory set as it needs to be always present. FileAR.Change is only available in the mandatory access condition set as it can only be available once for each file. The structure of mandatory and optional access condition sets can be seen in [Figure 19](#).

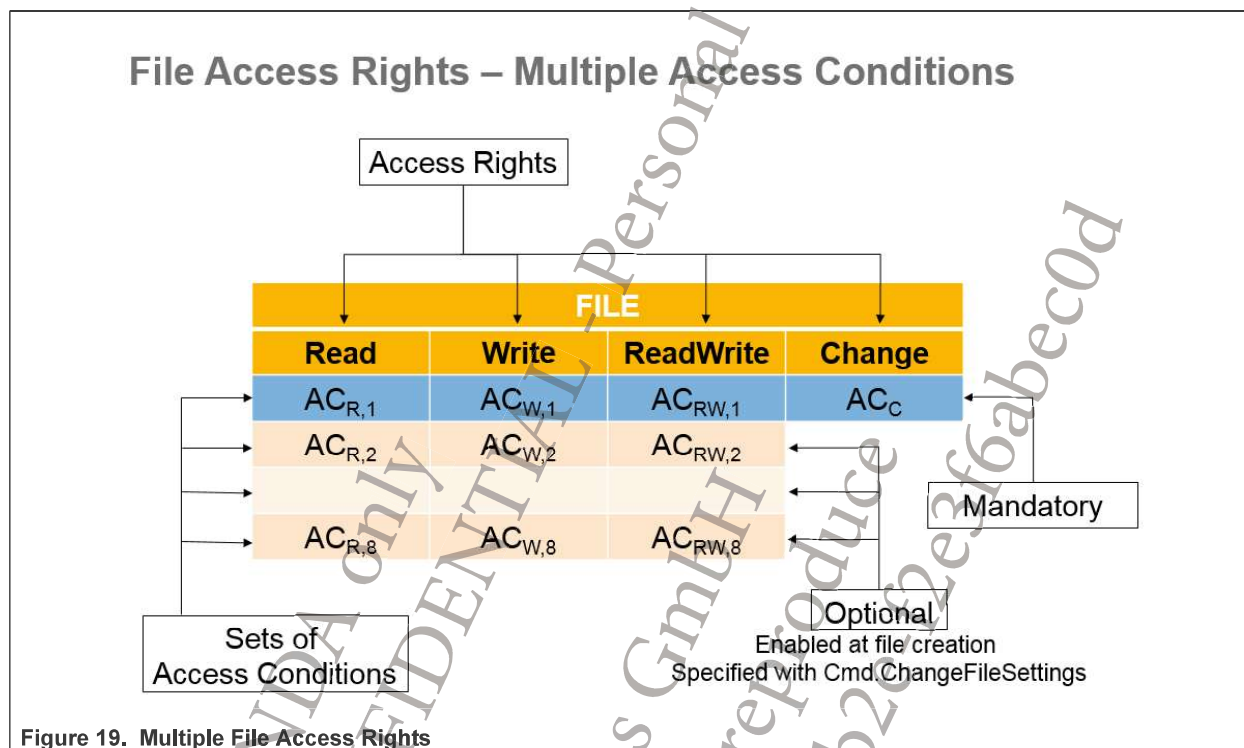


Figure 19. Multiple File Access Rights

Note:

The access rights management of a *FileType.TransactionMAC* file differs from the general access rights management for the other file types. For this special file type, the *FileAR.Write* is always set to 0xF, so writing is never possible and the *FileAR.ReadWrite* contains the key number of the *KeyID.AppCommitReaderIDKey* or 0xF in case the committing of the ReaderID is not required.

FileAR.Read and *FileAR.Change* are treated in the same way as for all other file types.

10.1.1 Enabling of File Access Conditions

The mandatory set of access conditions needs to be specified at file creation. If they need to be adapted later on, the *Cmd.ChangeFileSettings* can be used in order to change them. Another parameter, which specifies whether the file supports only one or multiple access condition sets, also needs to be specified during file creation.

If the number of access condition sets is limited to 1 during the file creation, it cannot be extended later on any more and the file cannot make use of multiple access condition sets.

If the number of access condition sets is not limited to 1, then the additional sets of access conditions can be specified with *Cmd.ChangeFileSettings*. This needs to be done because only the mandatory set is initialized during file creation, but the other access condition sets still need to be set.

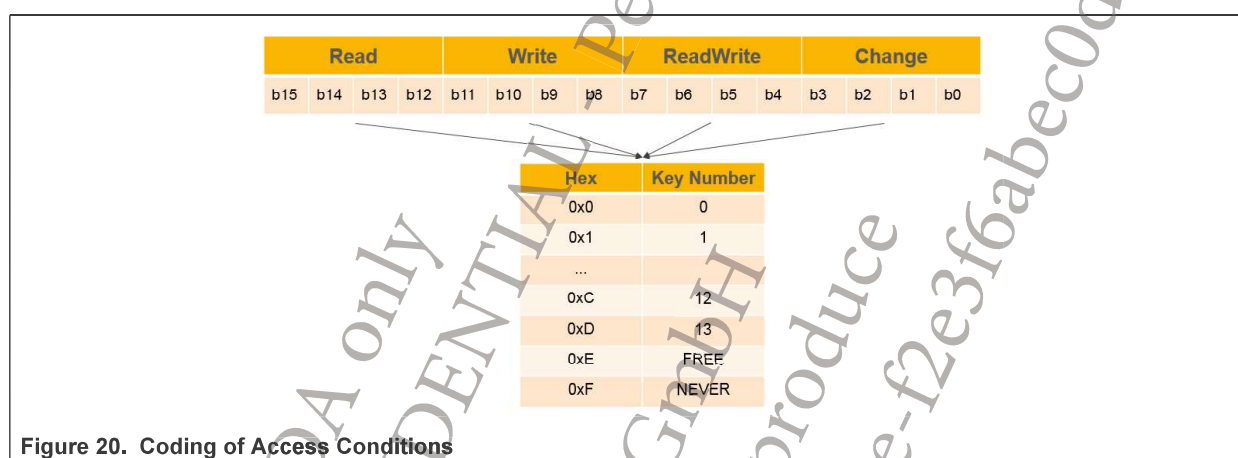
10.1.1.1 Different Kinds of Access Conditions

Each access condition is associated with an access right and evaluated to decide whether access to the file can be granted or not.

There are three kinds of access conditions:

- Authentication with a specified **KeyID.AppKey** is needed (0x00 – 0x0D) or authentication with an ECC Key where the CARootKey grants this access condition (and optionally the targeted certificate)
- Free access (0x0E)
- No access / Access forbidden (0x0F)

One set of access conditions is coded on 2 bytes, with each access condition in the set requiring 4 bits of space. The coding of an access condition set can be seen in [Figure 20](#).



10.1.1.2 Example: Setting of File Access Rights (mandatory Access Condition Set) during File Creation

FileNo = 01

ISOFileID = 2233

FileOption = additional access rights disabled, CommMode.Plain = 00

AccessRights = Read: 0x03, Write: 0x01, ReadWrite: 0x01, Change: 0x02

FileSize = 200000

In this example a backup data file with file number 01 and ISOFileID 2233 is created. It is configured in communication mode plain and only possesses one set of access conditions, that's why the FileOption is set to 00. The four access conditions are given and initialized directly during file creation, with key number 3 being the ReadKey, key number 1 being the WriteKey as well as the ReadWriteKey and key number 2 being the ChangeKey. The detailed APDU exchange can be seen in [Table 56](#).

Table 56. Setting of File Access Rights (mandatory Access Condition Set) during File Creation

Step	Command		Data Message
1	Cmd.CreateBackupDataFile C-APDU	>	CB012233003112200000
2	R-APDU	<	00

10.1.1.3 Example: Setting of File Access Rights (multiple Access Condition Sets) during File Creation

FileNo = 02

ISOFileID = 4455

FileOption = additional access rights enabled, CommMode.Plain = 80

AccessRights = Read: 0x03, Write: 0x01, ReadWrite: 0x02, Change: 0x03

FileSize = 300000

In this example a backup data file with file number 02 and ISOFileID 4455 is created. It is configured in communication mode plain and has the additional access conditions enabled, that's why the FileOption is set to 80. The four access conditions are given (the mandatory set) and initialized directly during file creation, with key number 2 being the ReadKey, key number 3 being the WriteKey as well as the ReadWriteKey and key number 1 being the ChangeKey. The detailed APDU exchange can be seen in [Table 57](#).

Table 57. Setting of File Access Rights (multiple Access Condition Sets) during File Creation

Step	Command		Data Message
1	Cmd.CreateBackupDataFile C-APDU	>	CB024455802331300000
2	R-APDU	<	00

10.1.2 Changing of File Access Conditions

In order to change one of the access condition sets or to initialize the additional access condition sets (which are empty after file creation), the Cmd.ChangeFileSettings needs to be used.

When using this command, the mandatory set of access conditions always is updated, as it includes this parameter as mandatory. Additionally the number of additionally access rights needs to be specified, following by the actual values of the additional access right sets. The detailed description of the command can be found in Ref. [\[1\]](#).

10.1.2.1 Example: Setting of Additional Access Rights by using Cmd.ChangeFileSettings

This example is the continuation of example [Section 10.1.1.3](#). After the file was created with additional access rights enabled, they will be initialized now with the following values.

Mandatory access right set (was already specified during file creation): Read: 0x02, Write: 0x03, ReadWrite: 0x03, Change: 0x01

Additional access right set 1: Read: 0x01, Write: 0x03, ReadWrite: 0x00, Change: 0x0F

Additional access right set 2: Read: 0x03, Write: 0x02, ReadWrite: 0x0F, Change: 0x0F

Additional access right set 3: Read: 0x00, Write: 0x00, ReadWrite: 0x00, Change: 0x0F

Table 58. Setting of Additional File Access Rights using Cmd.ChangeFileSettings

Step	Command		Data Message
1	Cmd.GetFileSettings C-APDU FileNo 02	>	F502
2	R-APDU	<	000180233130000000
3	Cmd.ChangeFileSettings Plain	=	5F02802331030F13FF320F00
4	Cmd.ChangeFileSettings enc	>	5F01[encrypted input]
5	R-APDU	<	00

10.1.2.2 Example: Multiple Access Condition Scenario

This example relies on the following two access right sets:

Mandatory access right set: Read: 0x01, Write: 0x0E, ReadWrite: 0x01, Change: 0x0F

Additional access right set 1: Read: 0x01, Write: 0x02, ReadWrite: 0x02, Change: 0x0F

Additionally to these access right sets the file is configured to use the fully encrypted communication mode.

This scenario is a bit more complex to understand as in the mandatory set the Write access condition is set to free, whereas in the additional access right set it is set to KeyNo 0x02.

What this means for the card user:

- When no authentication is established, the file can be written in plain
- When an authentication with a key different to KeyNo 0x02 is established, the file can be written in plain
- When an authentication with KeyNo 0x02 is established the file needs to be written fully encrypted

10.1.3 File Access Conditions related to SDM

For MIFARE DUOX the described access rights and access condition sets, as outlined in [Section 10.1.1](#) and [Section 10.1.2](#), are existing in the same way, but have been extended with access conditions reflecting the access rights that are needed for Secure Dynamic Messaging (SDM).

Once SDM is enabled for a Standard Data File, additionally 3 access rights can be defined:

- SDM Meta Read
- SDM File Read
- SDM Counter Retrieval

The SDM related access rights can only be set, if SDM was enabled for the Standard Data File during file creation, and if SDM is activated for the file. For all three mentioned access rights, an available application key number or the FREE (0xE) or NEVER (0xF) access right can be set.

The purpose / functionality of the three mentioned access rights is the following:

- SDM Meta Read - Encryption of the PICCData (metadata) using the specified application key number
 - Any available app key - Encrypted PICCData mirroring using the specified application key
 - 0xE - Plain PICCData mirroring
 - 0xF - No PICCData mirroring
- SDM File Read - Encryption of the mirrored file data using the specified application key number in non-authenticated state
 - Any available ap key - Free access to ReadData and ISOReadBinary commands, but encrypted SDMFileData mirroring using the specified application key
 - 0xE - Not defined
 - 0xF - No SDM for
- SDM Counter Retrieval - Possibility to retrieve the associated SDMReadCtr using the GetFileCounters command after an authentication with the specified application key number
 - Any available ap key - authentication with the specified application key require for being able to send the GetFileCounters command
 - 0xE - Free access
 - 0xF - No access, reading out SDMReadCtr is forbidden

Table 59. Command List associated to the additional SDM Access Rights

SDMMetaRead	SDMFileRead	SDMCtrRet
-	Cmd.ReadData	Cmd.GetFileCounters
	Cmd.ISOReadBinary	

How the SDM related file settings can be enabled and changed is explained in detail in [Section 4.2](#).

11 Data Management

MIFARE DUOX maintains user data in files of specific types. The user can access and manage the data through functions specific to file type. Access to data inside the files is limited by the access rights which are set at file level during file creation.

For all file types besides `FileType.StandardData`, the backup mechanism has to be considered when accessing or changing file data. This transaction oriented backup mechanism ensures, that data that has been written or updated in a file that supports the backup mechanism, will be visible only after a `Cmd.CommitTransaction` has been successfully executed. The transaction mechanism is described in [Section 12](#) in detail.

The detailed description of all the data management commands can be found in Ref. [\[1\]](#).

11.1 Example: Using the Linear Record File

Following example shows how to construct a data manipulation command by using the `WriteRecord` command in `CommMode.Full`.

Table 60. Write a Record into a LinearRecordFile

Step	Action	Direction	Command	Comment
1	AuthenticateEv2First Part 1	>	710000	
2	Response Part 1	<	AFF545FBD2F3BC4364D825B25 C781C24E1	
3	AuthenticateEv2First Part 2	>	AF79B1F4D6B55EAD6F1DF3 D0B01EB1DADFC19866235187 CBC1CFC84F6A2D151E	
4	Response Part 2	<	0021DB934655D39DE0 E580603223EB879268A7AAA41F9 B169F619C7DE20A2AD3C2	
5	K _{Ses} ENC	=	151D2E58CC8338133069F27 CDC873AFB	Session Key used for data encryption
6	K _{Ses} MAC	=	E602CEF77673CB45F1629A48378 F37BB	Session Key used for data CMACing
7	TI	=	388A4EDC	Transaction Identifier
8	Data to Write	=	00112233445566778899 AABBCCDDEEFF000102030405 060708090A0B0C0D0E0F	Target is a LinearRecordFile with a RecordLength of 32 byte (0x20). The file demands CommMode.Full
9	EncryptionIV = E(K _{Ses} ENC, A55A TI 00000000000000000000)	=	F4C9A4267C4345193E7CD816C6 DE4BA6	IV for command data encryption
10	Plain data for encryption	=	00112233445566778899 AABBCCDDEEFF000102030405 060708090A0B0C0D0E0 F800000000000000000000 0000000000000000	Record data with length 32 appended with 80h and then padded to the next multiple of the AES Block Size
11	Encrypted data	=	8C35A3DF3287B2B3B8CBB27D29 C05940580AFBBEB15C4E12209 A86CA8B85757319F9C79E53 D982833A0C12AF7ED3FE34	Encrypted data which is sent to the PICC

Table 60. Write a Record into a LinearRecordFile...continued

Step	Action	Direction	Command	Comment
12	MAC Input Data = Cmd CmdCtr TI Cmd Header E(CmdData)	=	3B0000388A4 EDC100000002000008C35 A3DF3287B2B3B8CBB27D29 C05940580AFBBEB15C4E12209 A86CA8B85757319F9C79E53 D982833A0C12AF7ED3FE34	MAC input, 3B is the command code for WriteRecord, the CmdCtr is 0x0000 here, as its the first command sent and the Header is FileNumber (0x10), the Offset (0x000000) and the record length (0x000020)
13	CMAC	=	EEB1A51654C016F86C35A69A934 ECE22	CMAC calculated with K _{SesMAC}
14	WriteRecord	>	3B100000002000008C35A3 DF3287B2B3B8CBB27D29 C05940580AFBBEB15C4E12209 A86CA8B85757319F9C79E53 D982833A0C12AF7ED3FE34B116 C0F8359A4E22	Command sent to the PICC
15	Response + CMAC	<	006A66DD64858F6F62	The received Response including CMAC
16	MAC input data	=	000100388A4EDC	The MAC Input data contains the success code (0x00), the CmdCtr which is now 0x0001 and the TI
17	CMAC	=	446A7C6658DD5B644A853B8F436 FE862	The calculated CMAC is truncated using every second byte. It fits the received one.

12 Transaction Management

MIFARE DUOX supports a mechanism to group all the data management commands which are executed into one single transaction. This means, that one transaction can consist of multiple read and write commands to different files inside one application (or inside two applications when using the shared application management). A transaction needs to be committed, e.g. set to valid, using the [Cmd.CommitTransaction](#). Only after committing the transaction, the changes get active in the files. Before committing, the file content reflects the old/original content, how it was before a write operation took place.

This transaction oriented mechanism also supports committing or aborting all write operations to files (with integrated backup mechanisms) via a single command. Excluded from the transaction mechanism is the [FileType.StandardDataFile](#). [Cmd.AbortTransaction](#) sets all write operations that were executed on files to invalid, and ensures that the file content does not change, and the original content stays valid.

In order to start a transaction, there is no dedicated execution of a special command needed. A transaction is started automatically as soon as an application (or two applications, when using shared application management) is selected using [Cmd.SelectApplication](#). Also after committing the previous transaction, a new transaction is immediately automatically started. A transaction is finished when executing the successful [Cmd.CommitTransaction](#) or [Cmd.AbortTransaction](#).

MIFARE DUOX also supports the so called Transaction MAC and/or Transaction SIG feature. This feature allows a third party to check the authenticity of a transaction and to detect fraudulent scenarios. A detailed description can be found in the following sections, starting with [Section 12.1](#) and [Transaction SIG](#)

12.1 Transaction MAC

In many application scenarios, payment terminals (point of services) are run by third party merchants and not by the payment providing company or the application provider directly.

Transactions which are done using the POS at the merchant by the customer (using the customer PICCs) get reimbursed later by the clearing house or the backend of the PICC issuer respectively the application provider.

This setup allows multiple opportunities for merchants insert frauds. Examples for fraudulent scenarios are:

1. PICC was never used, but merchant claims it to the backend
2. Merchant claims higher amount for reimbursement than was actually debited from the PICC
3. Data gets manipulated differently than stated in a contract between merchant and backend provider
4. Merchant claims multiple reimbursements for the same transaction

All these fraudulent activities can be detected by the Transaction MAC feature of MIFARE DUOX.

Using the Transaction MAC feature, the PICC generates a MAC over all data manipulation commands that happened during one transaction, by using a key that is only known by the PICC and backend ([KeyID.TransactionMACKey](#)).

The PICC also keeps a counter (Transaction MAC Counter = TMC), which is increased after every successful transaction.

Depending on the configuration of the PICC, the merchant can retrieve the encrypted MAC (Transaction MAC Value = TMV) as well as the Transaction MAC Counter. If configured, he can also be forced to commit the current Reader ID (the ID of the terminal where the transaction happened) and retrieve the encrypted previous Reader ID. All the encrypted values that the merchant can get from the card need to be reported to the backend in order to validate the parameters there and get a reimbursement for the successful transaction. The interaction between the merchant, the customer and the backend can be seen in [Figure 21](#).

12.1.1 Committing the Reader ID

If there are multiple merchants running installations for the same application, sometimes it is the case that the fraudulent transaction is detected by our TMAC functionality but it cannot be related to the scamming merchant. To solve this issue, an application can be configured (optional) in such a way that the merchant must commit the ID of its terminal (ReaderID) using `Cmd.CommitReaderID` before doing the transaction finalization using `Cmd.CommitTransaction`. The ReaderID is called TMRI - Transaction MAC ReaderID and is 16 bytes long. Committing this ReaderID requires a prior authentication with an application key assigned for this purpose while activating the Transaction MAC feature (`KeyID.AppCommitReaderID`).

Typically the `Cmd.CommitReaderID` will require a SAM inside the reader, so that the ReaderID cannot be forged or attacked by a fraudulent attacker. The ReaderID then should be the SAM UID, or a unique number that is stored inside the SAM of the reader. Also the `KeyID.AppCommitReaderID` should be stored in the SAM securely.

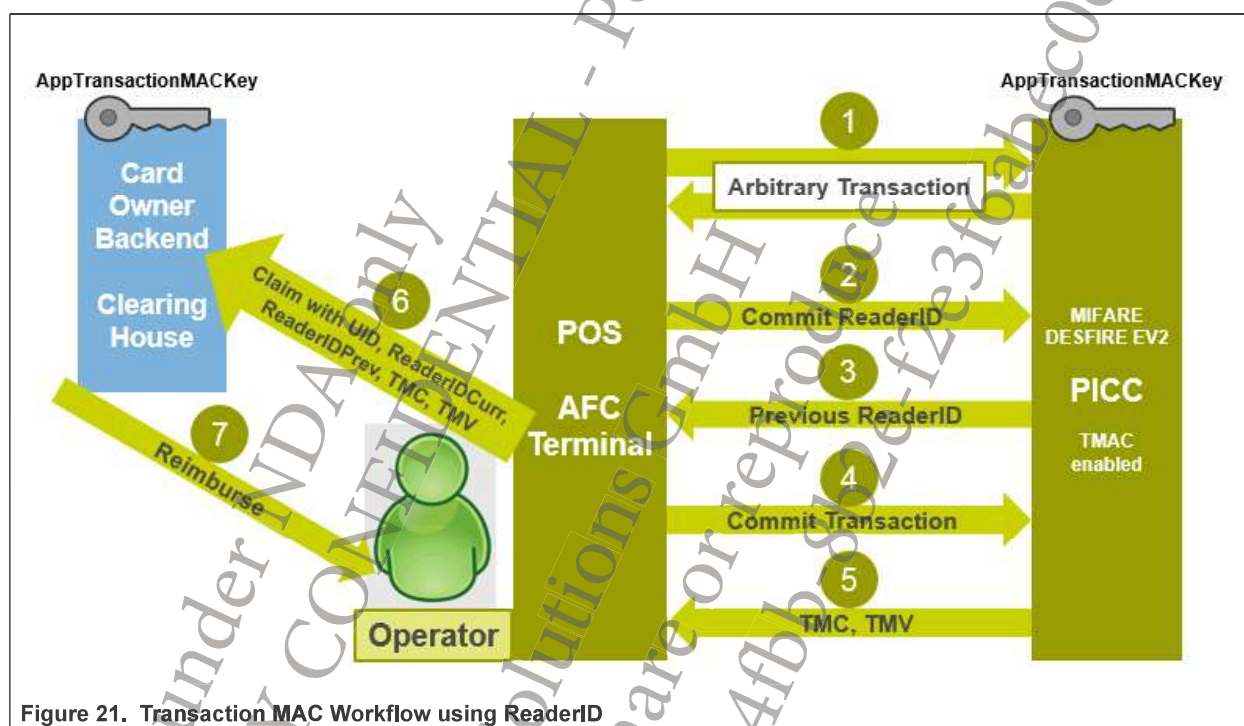


Figure 21. Transaction MAC Workflow using ReaderID

12.1.2 Transaction MAC Enabling

The TMAC feature in MIFARE DUOX is enabled automatically when creating a Transaction MAC File (`FileType.TransactionMAC`). This TMAC file contains the `KeyID.AppTransactionMACKey` as well as the 4 byte Transaction MAC Counter (TMC) and the 8 byte Transaction MAC Value (TMV).

The TMC starts from 0x00000000. After each successful commit transaction, the counter is increased by 1. If the TMC reaches 0xFFFFFFFF, no more transactions can be executed.

The TMV is the 8 byte CMAC computed over the Transaction MAC Input (TMI), which consists of the command data and other information of all executed commands during one transaction.

The Transaction MAC file also internally holds the `TMRIPrev` (the ReaderID of the previous transaction) as well as the `KeyID.AppTransactionMACKey`. The `KeyID.AppTransactionMACKey` needs to be provided as a parameter when executing `Cmd.CreateTMACFile` (Examples for TMAC File creation can be seen in [Section 12.1.3](#)).

12.1.3 Create Transaction MAC File Command

Cmd = CE

FileNo = 06

FileOption = 03 (Fully encrypted)

AccessRights = 301F (LSB first. Read Key 1, **Write access will be always F**, KeyID.AppCommitReaderIDKey 3, Configuration Change Key 0)

TMKeyOption = 02 (**Only AES is supported**)

TMKey = A0B0C0D0E0F000102030405060708090

TMKeyVersion = 00

Total Command = CE0603301F02A0B0C0D0E0F00010203040506070809000

Please note that previous reader id (TMRIPrev) is not provided at Cmd.CreateTransactionMACFile. The initial TMRIPrev is set to all zero bytes (16 bytes)

12.1.3.1 Example: Create Transaction MAC File in EV2 Secure Messaging

KeyID.SesAuthENCKey = 279ABB613B4C873457FDB3CE796CB7A6 (generated from the current authentication).

KeyID.SesAuthMACKey = 213FD7FABC970569CB12A725F05C1D2C (generated from the current authentication).

Current CmdCtr = 0200 (LSB first)

TI = 2EE822E8

Table 61. Create Transaction MAC File Example (in EV2 Secure Messaging)

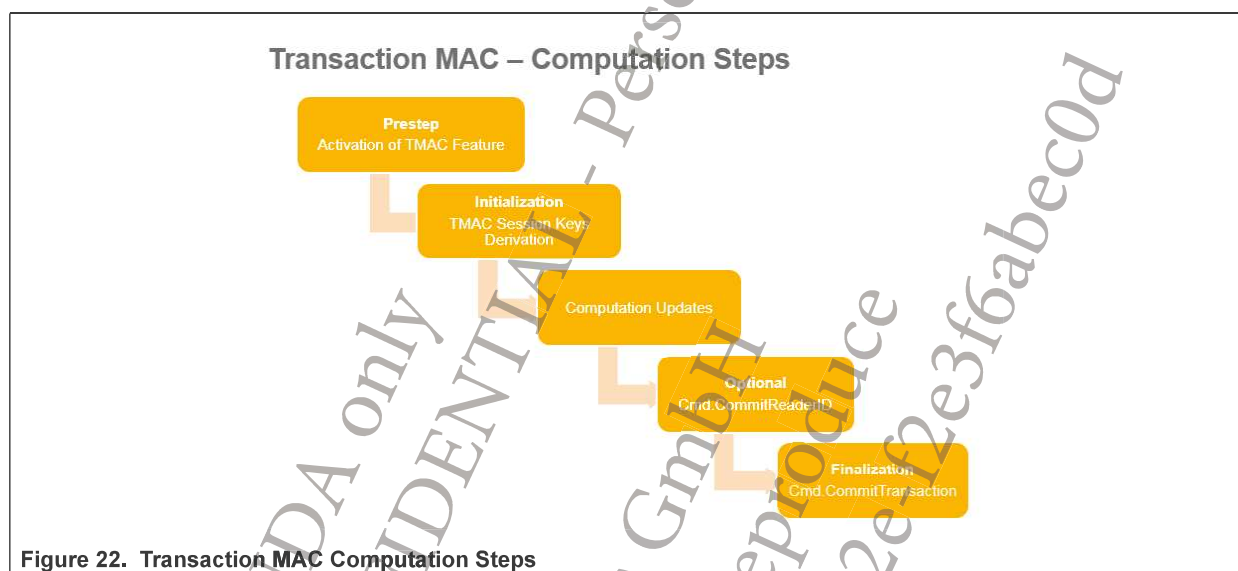
Step	Command	Data Message
1	Data to be encrypted = Command Data (TMKey TMKeyVersion)	= A0B0C0D0E0F00010203040506070809000
2	Padded to make multiple of 16 bytes (0x80 followed by 00s)	A0B0C0D0E0F00010203040506070809000800000 00000000000000000000000000000000
3	IVc = E(KSesAuthENC, 0xA5 0x5A TI C mdCtr 0000000000000000)	= B83AAA959FDF012705342696290A4704
4	Standard Encryption using the KSesAuth ENC	= F412B1F610F6DCD9001E24DFD82974C9C885B3D4F322482 DAE299672BB43A657
5	Data for calculating CMAC (Cmd CmdCtr TI FileNo Access Rights TMKeyOption Enc(CmdData))	= CE02002EE822E80603301F02F412B1F610F6DCD9001E24 DFD82974C9C885B3D4F322482DAE299672BB43A657
6	IV for CMAC calculation	00000000000000000000000000000000
7	CMAC	= 4436555C7ED5E1B1
8	Cmd.CreateTransactionMACFile C-APDU	> CE0603301F02F412B1F610F6DCD9001E24DFD82974C9C885 B3D4F322482DAE299672BB43A6574436555C7ED5E1B1
9	Updated CmdCtr	= 0300
10	R-APDU	< 008289DA269D6F7D2F
11	Data for calculating CMAC on response (RC CmdCtr TI)	= 0003002EE822E8

Table 61. Create Transaction MAC File Example (in EV2 Secure Messaging)...continued

Step	Command		Data Message
12	CMAC to be received	=	8289DA269D6F7D2F
13	Compare CMAC calculated and received from the card	=	8289DA269D6F7D2F ? 8289DA269D6F7D2F

12.1.4 Transaction MAC Computation Steps

The Transaction MAC computation requires the execution of multiple sequential steps (see [Figure 22](#)).



12.1.4.1 Transaction MAC Computation – Prestep

The activation of the TMAC feature is done by creating the Transaction MAC file as already described in previous sections.

12.1.4.2 Transaction MAC Computation – Initialization

On transaction start (before performing any commands that need a Transaction MAC computation update):

- TMI is set to an empty string
- TMRICur is set to an empty string
- Two TMAC session keys are derived (from KeyID.AppTransactionMACKey)
 1. KeyID.SesTMMACKey for computing TMV
It is based on a session vector SV1:
SV1 = 0x5A || 0x00 || 0x01 || 0x00 || 0x80 || (TMC + 1) || UID || Padding (if necessary to make multiple of 16)
KeyID.SesTMMACKey = CMAC* calculation from (KeyID.AppTransactionMACKey, SV1)
 2. KeyID.SesTMENCKey for encrypting previous ReaderID (if used)
It is based on a session vector SV2:
SV2 = 0xA5 || 0x00 || 0x01 || 0x00 || 0x80 || (TMC + 1) || UID || Padding (if necessary to make multiple of 16)
KeyID.SesTMENCKey = CMAC* calculation from (KeyID.AppTransactionMACKey, SV2)
*CMAC algorithm described in NIST Special Publication 800-38B (see Ref. [\[1\]](#))

12.1.4.2.1 Example: Generation of TMAC Session Keys

KeyID.AppTransactionMACKey = A0B0C0D0E0F000102030405060708090

UID = 04175D87000000 (7-byte UID, so no padding is needed)

Current TMC = 05000000 (4 bytes, LSB first)

SV1 = 5A0001008006000000004175D87000000

KeyID.SesTMMACKey = 237D87D5C2789702B18CB29675AEF3C6

SV2 = A50001008006000000004175D87000000

KeyID.SesTMENCKey = FF9C67FE242109C1AEB42A7CCC004C4E

12.1.4.3 Transaction MAC Computation – Update

The Transaction MAC Input (TMI) gets updated for each data manipulation command including `Cmd.ReadData`, `Cmd.GetValue`, `Cmd.ReadRecords` and much more. Please see Ref. [1] for all exact details, how the TMI gets updated for the different data manipulation commands (few examples are also given below)

12.1.4.3.1 Example: Update TMI

Debit value file by 1

TMI will be updated as TMI = TMI||Cmd||FileNo||Value|| ZeroPadding (10 byte padding to make it to multiple of 16)

Cmd: "DC", File ID: "03", Value: "01000000",

Current TMI is empty string as transaction is just initialized

New TMI=DC||03||01000000||000000000000000000000000 = DC030100000000000000000000000000

Write a record in record file (cyclic)

TMI will be updated as TMI = TMI last TMI - already a multiple of 16 byte ||Cmd||FileNo||Offset||Length||Zero Padding(padding to make it multiple of 16 up to this parameter)||Data||ZeroPadding(again padding to make it multiple of 16)

Cmd: "3B", File ID: "04", Offset: "000000", Length: "0B0000", Record: "020709183530DE02006400",

Current TMI = DC030100000000000000000000000000

New TMI=DC030100000000000000000000000000 ||3B||04||000000||0B0000||0000000000000000||020709183530DE02006400||0000000000 = DC0301000000000000000000000000003B04000000B0000000000000000000020709183530DE020064000000000000

In some applications one might need to read the current content of a record file or value file, but does not want to have the command in the MAC calculation. In such cases an abort transaction command (`Cmd.AbortTransaction`) can be performed after execution of those commands to exclude those contents from the Transaction MAC calculation. There are also use cases where those commands shall be included. For example, if a value alike operation is implemented in data files, the backend might need to know the current value as well as the updated value. Or if just some data is read (e.g. card holder ID) and reimbursement is done with a fixed value based on this read out ID. Please note that execution of `Cmd.AbortTransaction` command doesn't lose the authentication state, but the transaction is initiated again.

12.1.4.4 Transaction MAC Computation – CommitReaderID

If the execution of `Cmd.CommitReaderID` is a must, an authentication with `KeyID.AppCommitReaderIDKey` has to be performed. After successful authentication, `Cmd.CommitReaderID` is executed, which provides the current `ReaderID` and receives the encrypted previous `ReaderID`. Encryption is done using the AES block cipher according to CBC mode of NIST SP800-38A without any padding (IV is the zero-bytes IV). After successful execution of `Cmd.CommitReaderID`, the TMI gets updated as follows:

12.1.4.4.1 Example: Execution of CommitReaderID Command

TMI will be updated as $TMI = TMI \parallel Cmd \parallel TMRICur \parallel EncTMRICur \parallel ZeroPadding$ (to make it a multiple of 16 bytes)

Cmd = C8

TMRIC_{ur} = 04123B01D52880A1A2A3A4A5A6A7A8A9

In EV2 SM

Authenticated with Cmd.AuthenticateEV2NonFirst using KeyID.AppCommitReaderIDKey

KeyID.SesAuthMACKey = BFC148020D639B8358FBF92F1297D6F4

TI = 1DD6C464

CmdCtr = 0200

Table 62. Example: Commit Reader ID (in EV2 Secure Messaging)

Step	Command		Data Message
1	Data to calculate CMAC (Cmd CmdCtr TI TMRI)	=	C802001DD6C46404123B01D52880A1A2A3A4A5A6A7A8A9
2	CMAC	=	4380E238D3A7ADCE
3	Cmd.CommitReaderID C-APDU	>	C804123B01D52880A1A2A3A4A5A6A7A8A94380E238D3A7 ADC
	Updated CmdCtr	=	0300
4	R-APDU	<	00710514F847DF5244F2D22D7D64BC0A145CDCB7DA885 DFC1D
5	EncTMRI	=	710514F847DF5244F2D22D7D64BC0A14 Please see Encryption of Transaction MAC ReaderID below

12.1.4.4.2 Example: Encryption of Transaction MAC ReaderID

KeyID.SesTMENCKey = FF9C67FE242109C1AEB42A7CCC004C4E

TMRIprey = 04123B01D52880A1A2A3A4A5A6A7A8A9 (same as the current one - as same TMRI has been used in previous transaction)

EncTMRI = 710514F847DF5244F2D22D7D64BC0A14

[illegible]

09183530DE02006400000000000000C804123B01D52880A1A2A3A4A5A6A7A8A97105

```
14F847DF5244F2D22D7D64BC0A1400000000000000000000000000000000
```

12.1.4.5 Transaction MAC Computation – Finalization

Cmd.CommitTransaction finalizes the transaction and therefore also the TMAC calculation. The TMV gets calculated (8 byte CMAC is calculated over the TMI using KeyID.SesTMMACKey according to EV2 Secure Messaging).

12.1.4.5.1 Example: Execution of Cmd.CommitTransaction and TMAC Finalization

[illegible]

KeyID.SesTMMACKey = 237D87D5C2789702B18CB29675AEF3C6

TI = 1DD6C464

CmdCtr = 0300

KeyID.SesAuthMACKey = BFC148020D639B8358FBF92F1297D6F4

Table 63. Example: Commit Transaction (in EV2 Secure Messaging)

Step	Command	Data Message
1	Data to calculate CMAC (Cmd CmdCtr TI Option)	= C703001DD6C46401
2	CMAC	= 19E80643AFFEA0DE
3	Cmd.CommitTransaction C-APDU	> C70119E80643AFFEA0DE
4	R-APDU	< 0006000000C6A2A823884CDE8A60B02FBAF6F9BE66
5	TMC	= 06000000
6	TMV	= C6A2A823884CDE8A

12.1.5 Retrieval of TMC and TMV

TMC and TMV can be retrieved as the response of Cmd.CommitTransaction (if bit 0 of the Option parameter is set to 1). TMC and TMV can be also read via Cmd.ReadData. Please note that Cmd.CommitTransaction returns TMC and TMV in plain.

In case TMC is considered as sensitive business information, one should use Cmd.ReadData and therefore the Transaction MAC File shall be created using CommMode.Full.

12.2 Transaction SIG

Similar as for the TMAC feature, also a Transaction Signature can be calculated instead of a CMAC. There is no difference in the input for the signature calculation, it reuses the input of the TMAC, the only difference is, that a specific ECC key which is allowed for use with TSIG needs to be present and specified. This is done in the `Cmd.CreateTransactionMacFile`.

In the following example, the PICCMasterKey as well as the AppMasterKey is 00000000000000000000000000000000

Table 64. Application creation and file creation for TSIG support

Step	Action	Direction	Command	Comment
1	AuthenticateEV2First Part1	>	710000	Authentication with PICCMasterKey.
2	Response 1	<	AFD1561739766FDD0D1 BDFED7659C103	
3	AuthenticateEV2First Part2	>	AF1C98CD1AAFD313E2020 C7833924180C9818E4CFF9A5 EAF1B5C2B951ED2D6EE23	
4	Response 2	<	00A3B2E9E4343D339982DF83B60 D98FE3ECA6773E05EE4C8D26F2 F7A3A5FAFA6D0	
5	Format	>	FCBF8DB237C0400409	Format the card
6	Success	<	00ABB28F279AE866E2	
7	CreateApplication	>	CA0100000F82AC644897A253A4 B2	Create the application. In this case, its AID 0x0000 01 There is no setting to be made at application creation to enable the TSIG feature
8	Success	<	00779748D0301E8AAE	
9	SelectApplication	>	5A010000	Select the just created application 000001.
10	Response	<	00	
11	AuthenticateEV2First Part1	>	710000	Authentication with App MasterKey.
12	Response 1	<	AFE5C14F196997EFCED66D86 CEB60DE8F5	
13	AuthenticateEV2First Part2	>	AF1000FF7525A45E108B50EE5 EA7B186FCE7883336D99492094 BF94FD776A78942	
14	Response 2	<	008F5CEE3D099ED5DC24 C46543970BD60EC156E2AD167 C2571D6723785F78329C4	
15	ManageKeyPair	>	4600000C40003000000000C82 F186757AB50DDDBDF6440538 EE9BFC9A2E103B4ACDA14	ManageKeyPair with option byte 0x00. This generates a ECC key in key slot 0x00 and responds the PubKey encrypted with the session keys. The key policy is set to 0x0040, which enables the key to be used with the TSIG feature.
16	Success + PubKey	<	00674971C7C00E06E8D8257E920 F185C88B5CE944F92CF942E4 B300C33240D8326BAB01EA11703 E1EA0EE03C484365FF05CF062 B666391D44F2ADC02DD8AD43 A13D24A8A4FFD606C8A3D7669 FC20D600E16B839E368347D33E	
16a	plain PubKey	=	04DF68234DBC710E800B256 E8905D88CF598244E2052E80 BAA31CD83A5CFC18451C2 D53343E6698F832D18222E1150 B1314890809C09534363FC2E56 DA35C6C3BC	
17	CreateStdDataFile	>	CD01010000000100610DD5F66 B854206	Create a StandardDataFile which will be later used for data operations
18	Success	<	00E0FC5B98E14F6981	

Table 64. Application creation and file creation for TSIG support...continued

Step	Action	Direction	Command	Comment
19	CreateTransactionMacFile	>	CE0203F01F80C720E5D61C5F68 C1E25799E8914638D61184CC57 BDA6DF85	Create a TransactionMac File. Access Rights are 0x1FF0 (Read with Key 1, Never Write, Never Read Write, Change with Key0). Write and ReadWrite are mandatory to be set to 0xF for TransactionMacFiles. As this example does not use the CommitReader ID function, there is no TMkey (used for ReaderID encryption) defined.
20	Response	<	00324F341FC0A70B31	

The following table is an example transaction with a read and a write command, that will result in a TSIG.

Table 65. Transaction with returned TSIG

Step	Action	Direction	Command	Comment
1	SelectApplication	>	5A010000	Select the application 000001.
2	Success	<	00	
3	AuthenticateEV2First Part1	>	710000	Authentication with AppMasterKey. Its not necessary here that this is the App MasterKey, it can be any application key that grants the needed Write/Read access rights.
4	Response 1	<	AF2B8C02EFE402025F513CE765D8F1FD10	
5	AuthenticateEV2First Part2	>	AF0B57780AA3387D14A75208FCAD071F3CF0B347 FB62144E0D4D54057CFF22B726	
6	Response 2	<	00AE0284498FD591A229153294465C3CB28B7858 CF07D1229890946C3D1F92D19D	
7	WriteData	>	3D0100000010000000112233445566778899 AABBCCDDEEFFB614AD4ED1B3E420	Write some data into the StdDataFile using CommMode-MACed
8	Success	<	00F10E38FC11C3C684	
9	ReadData	>	BD010000001000004440A071CA8B315A	Read some data back from the Std DataFile using CommMode-MACed
10	Success	<	0000112233445566778899AABBCCDDEEFF09C3D2 BD15142A5A	
11	CommitTransaction	>	C701BCC0A3BED046F9D8	Commit Transaction.
12	Success + TSIG	<	0001000000F7AD2AC46A3C4E8631FE6B1CC1 FE1C781EC87F6FB6AA2AD022AAF7BBBE4B8 777AFBADABE103BFE200F952D9EA70DD6B8037 4D7700A155EBF81E486E3C5B6678BCCDCF90 D732830A5	This will return the TSIG and the transaction MAC counter (TMC), which is for this example "01000000"

The actual transaction is now over. The card returned the transaction signature which now need to be verified by the backend. This might be necessary to ensure that a transaction was executed as intended and not manipulated by the reader in order to gain e.g. monetary advantages. As the private key is not present at the reader, the reader can not forge this signature, and therefore can not hide any additional or left out commands.

Table 66. TS|G verification

[illegible]

13 Proximity Check

The Proximity Check Architecture can be (but does not need to be) used in conjunction with the Virtual Card Architecture. It allows the PCD to verify whether the PD is within a certain distance. The goal is to avoid the following attacks:

- A relay attack where an attacker tries to use a PD that is not in proximity to set-up a transaction with a PCD. This attack can be undertaken with or without the support of the legitimate card holder.
- A relay attack where an attacker tries to obtain privacy sensitive information from a card holder. The attacker tries to find out whether the attacked PD contains a VC belonging to a certain installation by relaying the requests of a valid PCD of that installation to a PD under attack. If the PD is accessed by the PCD after a successful VC Selection procedure, the attacker knows the PD also belongs to the installation.

The working principle and the detailed command set as well as the benefits that the Proximity Check functionality offers are described in the MIFARE DUOX Data Sheet Ref. [\[1\]](#).

14 Originality Check

MIFARE DUOX supports an Originality Check Mechanism based on a card-unilateral asymmetric authentication, using an trust-provisioned ECC key and certificate from NXP. This key and certificate can be overwritten during card personalization and be used later in a user-specific infrastructure to determine if a card is belonging to this infrastructure or was provisioned by the correct card issuer.

The card-unilateral authentication is performed using the Cmd.ISOInternalAuthenticate. This command performs a 2-pass card-unilateral authentication as standardized in ISO/IEC 9798-3. Card-unilateral means, that during this authentication, the card proves its authenticity towards the reader, but the reader does not at all provide any information about its identity.

Table 67. Card-unilateral authentication for originality check purposes

Reader	Transmission	Card
Knows: Pub.Orig		Knows: Priv.Orig
Generates RndA		
Sends ISOInternalAuthenticate		
	[OptsA RndA >	generates RndB
		Sig.Orig = ECDSA _{Sign} (Priv.Orig, 0xF0F0 OptsA RndB RndA)
	< RndB Sig.Orig	Responds the Signature
Verifies Sig.Orig		

15 MIFARE DUOX APDUs

There are three types of MIFARE DUOX command-response APDU or framing methods existing:

- MIFARE DUOX Native APDU
- ISO/IEC7816 wrapping of MIFARE DUOX Native APDU
- Standard ISO/IEC 7816 APDU

The frame size that is currently supported by the MIFARE DUOX can be configured with each individual IC, offering a variety from FSCI = 0 up to FSCI = 8. This relates to a frame size of 16 bytes for FSCI = 0 up to a maximum frame size of 256 bytes for FSCI = 8.

The default FSCI for delivery of the MIFARE DUOX is set to FSCI = 5, which reflects an actual frame size of 64 bytes. This default frame size of 64 bytes is maintained for most of the native commands. A few command variants can make use of the larger FSC. These include: Cmd.ReadData, Cmd.WriteData, Cmd.ReadRecords, Cmd.WriteRecord and Cmd.UpdateRecord.

The exceptional commands listed here can make use of either native chaining (chaining frames using 0xAF) or the ISO/IEC 14443-4 chaining. An example of the ISO/IEC 14443-4 chaining mechanism can be seen in [Figure 24](#) and [Figure 25](#).

15.1 MIFARE DUOX Command – Response APDU (Framing)

All the APDUs (C-APDU and R-APDU) have to be embedded in the block transport protocol defined in ISO/IEC 14443-4, which can be seen in [Figure 23](#).

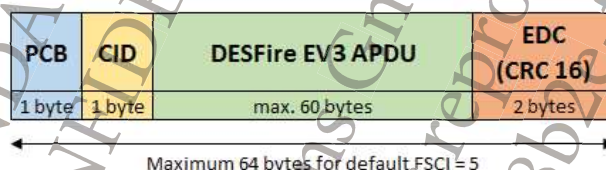


Figure 23. MIFARE DUOX APDU embedded into T=CL Frame

	Comment	Block No. (0)	PCD	PICC	Block No. (1)	Comment
1.	rule 1		$I(1)_0$	\Rightarrow	0	rule D
2.	rule B	1	\Leftarrow	$R(ACK)_0$		rule 2
3.	rule 7		$I(0)_1$	\Rightarrow	1	rule D
4.	rule B	0	\Leftarrow	$I(0)_1$		rule 10
5.			$I(0)_0$	\Rightarrow	0	rule D
6.	rule B	1	\Leftarrow	$I(0)_0$		rule 10

Figure 24. ISO/IEC 14443-4 Chaining on the PCD side

	Comment	Block No. (0)	PCD		PICC	Block No. (1)	Comment
1.	rule 1		I(0) ₀	⇒		0	rule D
2.	rule B	1		⇐	I(1) ₀		rule 10
3.	rule 2		R(ACK) ₁	⇒		1	rule E
4.	rule B	0		⇐	I(0) ₁		rule 13
5.			I(0) ₀	⇒		0	rule D
6.	rule B	1		⇐	I(0) ₀		rule 10

Figure 25. ISO/IEC 14443-4 Chaining on the PICC side

15.2 MIFARE DUOX Native APDUs

When using the MIFARE DUOX Native APDU format, the C-APDU is constructed in the following way: a concatenation of the command code (Cmd), the command header (Cmd Header) as well as the command data (Cmd Data) is embedded into the T = CL frame.

A command code is always required, but depending on the command either command header, command data or both can be missing as well.

The basic structure of a MIFARE DUOX Native C-APDU can be seen in [Figure 26](#).

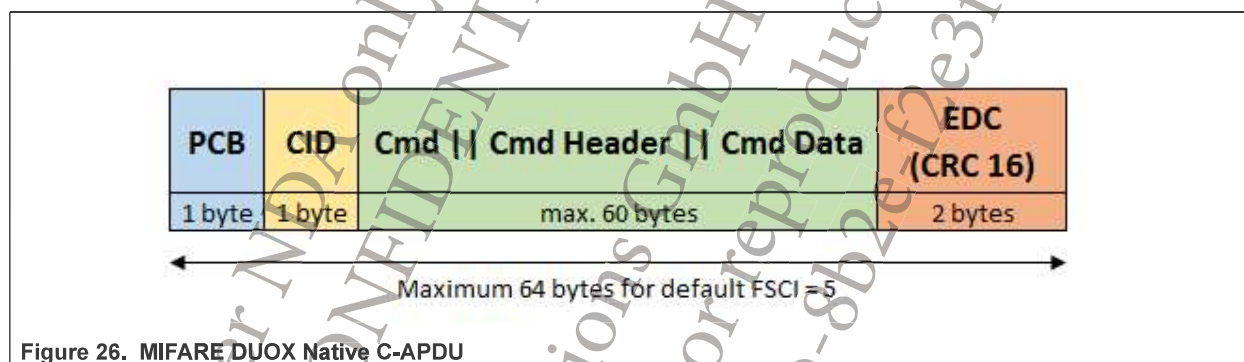


Figure 26. MIFARE DUOX Native C-APDU

The R-APDU is constructed in the following way: the response code is concatenated with the actual response data. A response code is always required in order to signalize the success or the failure reason of a command execution. The response data can be missing, as not all commands require the return of a response data.

The basic structure of a MIFARE DUOX Native R-APDU can be seen in [Figure 27](#).

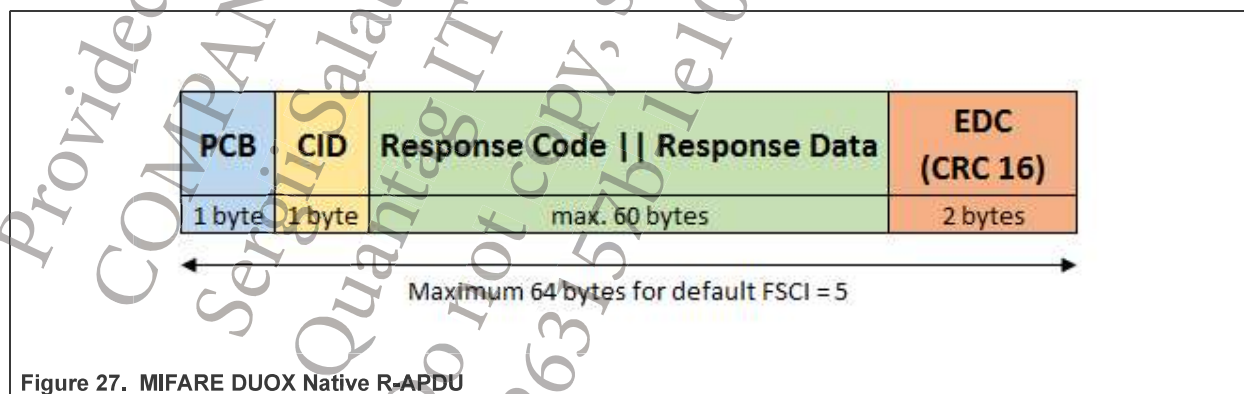


Figure 27. MIFARE DUOX Native R-APDU

15.3 ISO/IEC 7816 wrapping of MIFARE DUOX Native APDUs

In order to be compliant to the ISO/IEC 7816 standard, native MIFARE DUOX APDUs can be wrapped / converted into an ISO 7816 compliant APDU.

How the structure for the C-APDU and the R-APDU look like can be seen in [Figure 28](#) and [Figure 29](#).

Additionally also the extended length ISO/IEC 7816 APDUs are supported which require an adaption of the Lc field to 3 bytes as well as the Le field to 2 bytes.

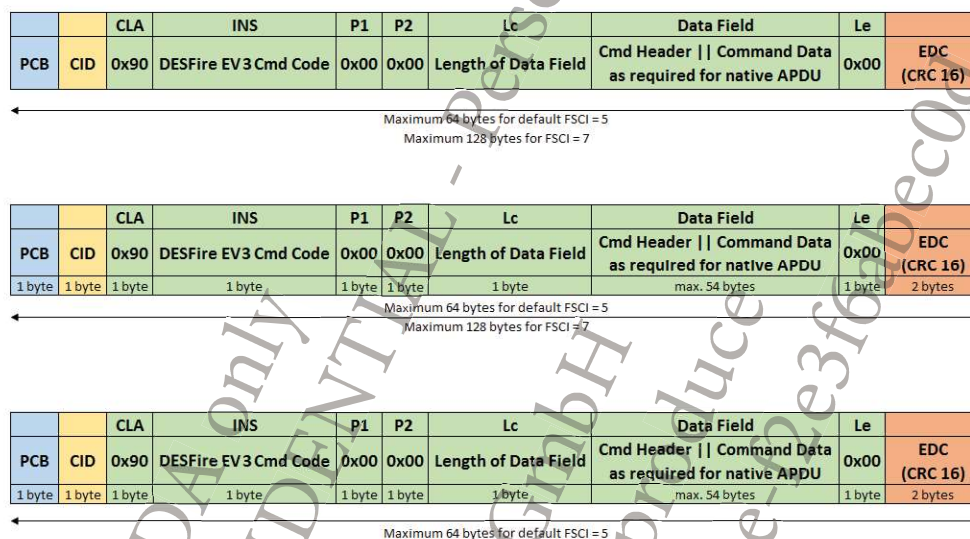


Figure 28. MIFARE DUOX Native C-APDU wrapped into ISO/IEC 7816 APDU format

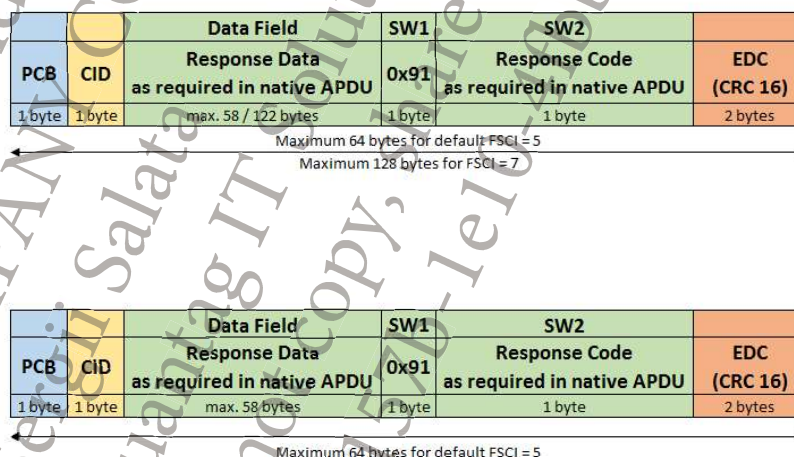


Figure 29. MIFARE DUOX Native R-APDU wrapped into ISO/IEC 7816 APDU format

15.3.1 Example: Cmd.ReadData using ISO/IEC 7816 APDU Wrapping

In [Figure 30](#) an example for Cmd.ReadData in the ISO/IEC wrapped APDU format which uses native chaining is shown.

In total 99 bytes will be read from file number 0x01, which requires a splitting of the read data into two frames. The first response frame delivers 58 bytes of the file data, the second response frame delivers 41 bytes of the file data, making 99 bytes in total.

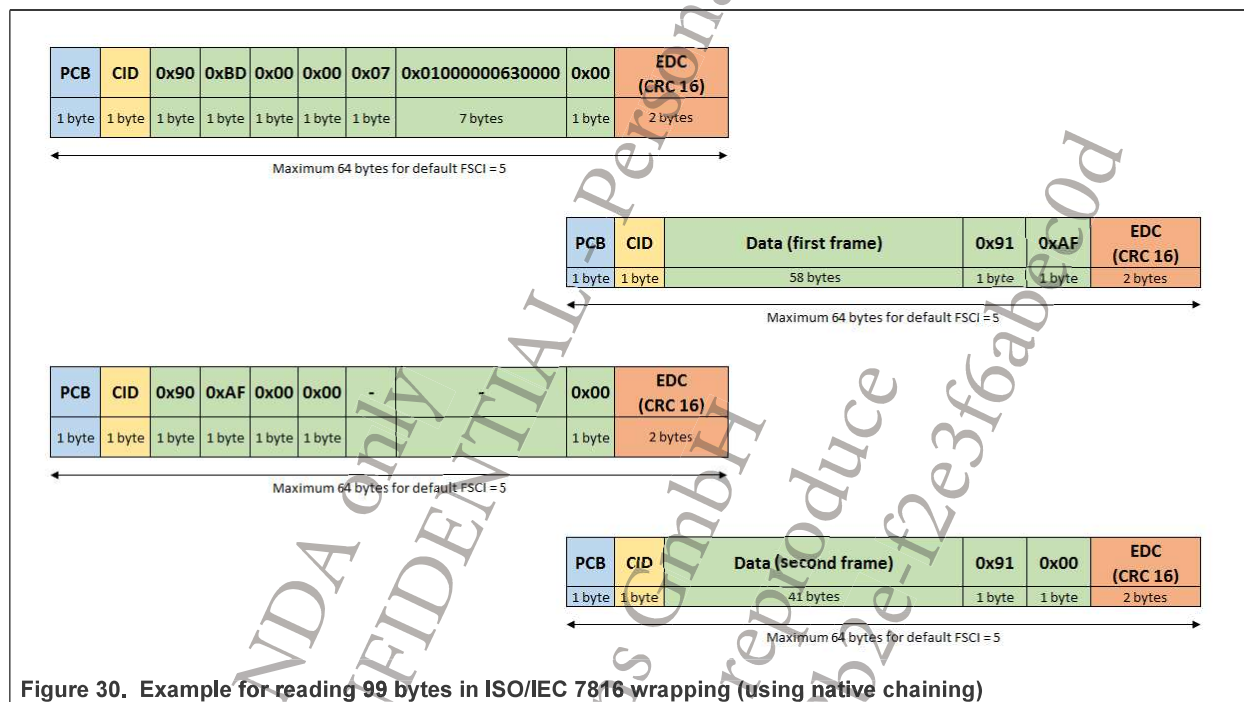
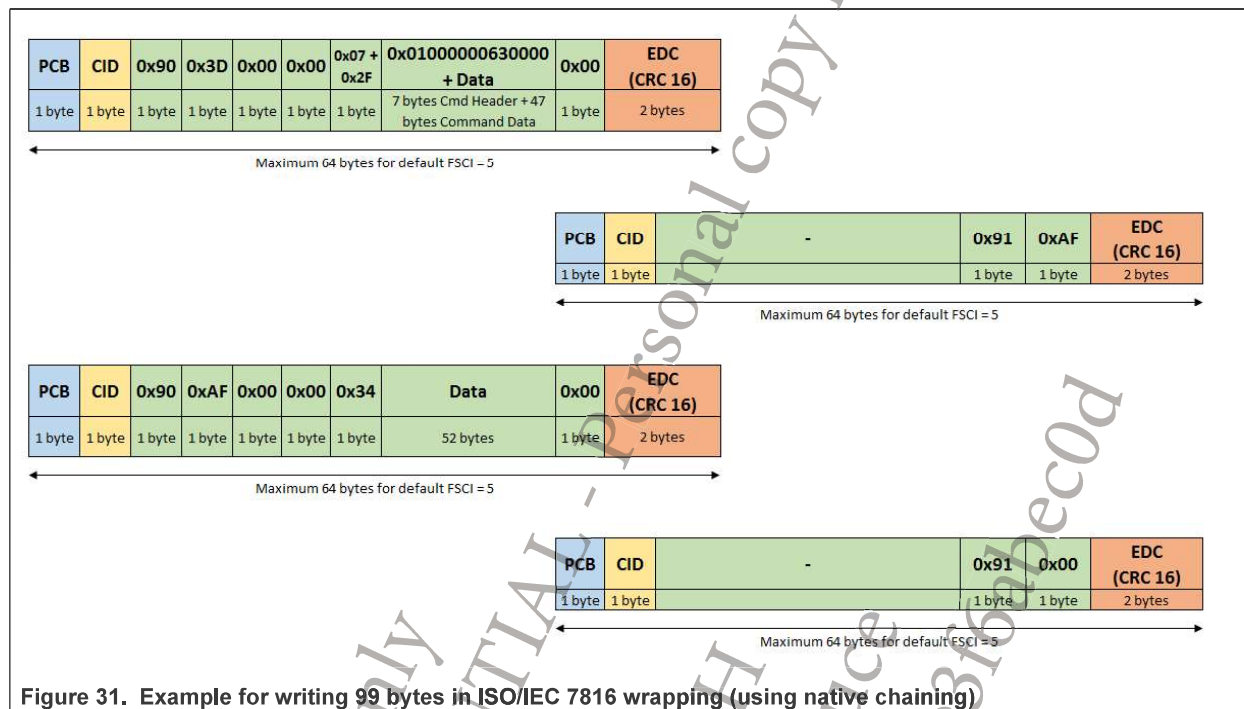


Figure 30. Example for reading 99 bytes in ISO/IEC 7816 wrapping (using native chaining)

15.3.2 Example: Cmd.WriteData using ISO/IEC 7816 APDU Wrapping

In [Figure 31](#) an example for Cmd.WriteData in the ISO/IEC wrapped APDU format which uses native chaining is shown.

In total 99 bytes will be written to file number 0x01, which requires a splitting of the data to be written into two frames. The first command frame delivers 47 data bytes, the second command frame delivers 52 data bytes, making 99 bytes in total.



15.4 Example: Cmd.WriteData – Writing to FileType.SdtDataFile using ISO/IEC 7816 Wrapping and CMAC communication

Secure Messaging Mode = EV2 Secure Messaging

Communication Type = CMAC

IV = 00000000000000000000000000000000

File no = 0x00

Offset = 000000

File Size = 100000 (16 bytes)

Data to Write = 00112233445566778899AABBCCDDEEFF

SessionENCKey = 349B77F969E354336764EAFBA03DFF3F

SessionMACKey = D81A077A64FD0D12A57DA95721D33022

Table 68.

Step	Command		Data Message
1	Input Data (native APDU without MAC)	=	3D0000000010000000112233445566778899AABBCCDDEEFF
2	Calculate CMAC on Input Data	=	60AD6D50327897F2
3	Native APDU	=	3D0000000010000000112233445566778899AABBCCDDEEFF60AD6D50327897F2
2	IV	=	00000000000000000000000000000000
3	Prepare data in wrapped APDU CLA INS (Native Cmd) P1 P2 Lc Data (file no offset length data CMAC) Le	=	903D00001F0000000010000000112233445566778899AABBCCDDEEFF60AD6D50327897F200
4	C-APDU	>	903D00001F0000000010000000112233445566778899AABBCCDDEEFF60AD6D50327897F200
5	R-APDU	<	00732201000C1A6B5A
5	Calculate CMAC on "00" (success code as in native)	=	732201000C1A6B5A
6	Updated IV	=	00000000000000000000000000000000
7	Compare CMAC calculated and received from the PICC	=	732201000C1A6B5A ? 732201000C1A6B5A

15.5 ISO/IEC 7816 Standard APDUs

The standard ISO/IEC 7816 APDUs are constructed as shown in [Figure 32](#) and [Figure 33](#).

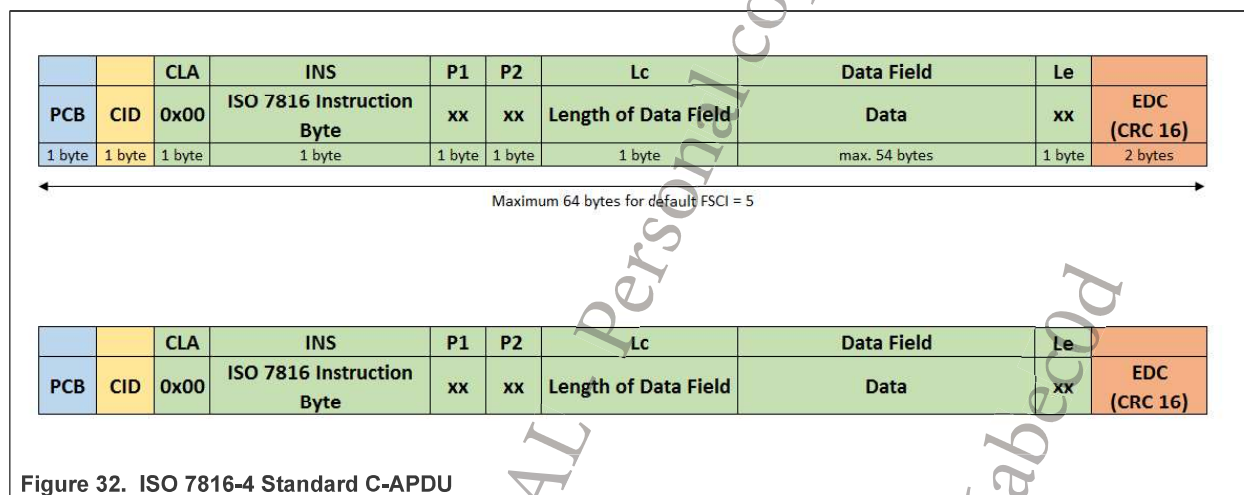


Figure 32. ISO 7816-4 Standard C-APDU

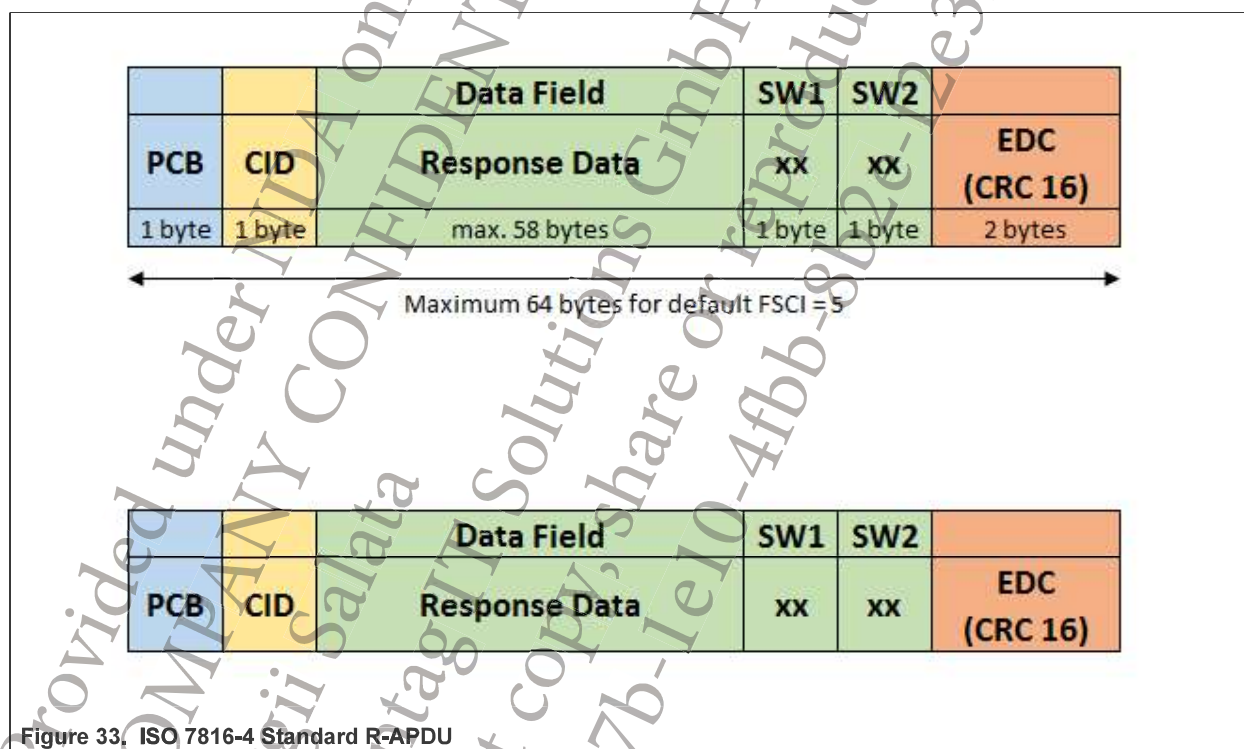


Figure 33. ISO 7816-4 Standard R-APDU

- ISO7816-4 INS as defined in Ref. [\[1\]](#).
- P1, P2: INS specific.
- Le if response expected, otherwise absent.
- SW1SW2: INS specific as documented in Ref. [\[1\]](#)
 - SW1SW2 = 9000, for successful command execution

16 Additional Hints and Recommendations

16.1 Key Management

KeyID.AppTransactionMACKey and KeyID.AppCommitReaderID should be UID diversified.

Cmd.CreateFile and Cmd.DeleteFile (for the different file types) shall be protected by KeyID.AppMasterKey (KeyID.AppMasterKey shall not be present at a merchant terminal). This will also make sure that KeyID.AppTransactionMACKey is not sent in plain (as there will be an active authentication) at Transaction MAC file creation. The TMAC file requires 3 blocks of memory and this memory will be reused in case the file is recreated. If a Transaction MAC file has to be deleted and recreated by the application owner, a new KeyID.AppTransactionMACKey should be used to avoid the overlap of the transaction MACs (TMC will be reset to zero at file creation). The counter that you should not only reset, that is, recreating the file using the same Transaction MAC Key. In these cases, TMI will be different for new transactions.

The system shall be designed in such a way that a merchant will not be able to forge the ReaderID (if used). A SAM shall be used for this purpose. The KeyID.AppCommitReaderID shall be stored in the SAM (dumping is not allowed). The SAM UID with added padding can be used as a ReaderID.

16.2 Application Management

Both in conventional and delegated application, one should initialize the default application keys (using Cmd.ChangeKeyEV2) before creating the file system.

Use KeyID.PICCDAMAuthKey as UID-diversified as this will avoid attacks on DAM commands of any PICC if the DAM keys of one PICC have been compromised.

Updating of the keys should be done in a secure environment if there is not enough mutual trust between application provider and PICC issuer.

When a delegated application is created targeting an already existing slot, the old application is deleted (in the background without the user noticing) before the new application is created. In case of tearing, this can lead to the situation that the old application is deleted without the new application being created. In such scenarios, Cmd.CreateDelegatedApplication needs to be executed again.

The PICC issuer must use real random numbers in the DAM initial application key encryption (EnCK KeyID.AppDAMDefaultKey). This will ensure that different application providers will receive different EnCK (encrypted default DAM application keys) even if they use identical KeyID.AppDAMDefaultKey. That means from the parameter of the Cmd.CreateDelegatedApplication one cannot distinguish that different application providers are using the same KeyID.AppDAMDefaultKey or an application provider is using the same key for different applications.

16.3 Initial trust of an ECC only application

MIFARE DUOX supports a mechanism to create an application without symmetric keys. A big advantage of this mechanism is, that no memory is reserved for symmetric keys, as the application will for the initial personalization use a temporary application master key (AppMasterTmpKey) that can be disabled after the application creation process, e.g. after the ECC keys were created. Further personalization can then be done using ECC mutual authentication.

The below example uses following keys:

Card Key Pair: Generated on MIFARE DUOX with Cmd.ManageKeyPair

Public key (returned from the card): 04D28C5A669113F1D0071F0FDE00EFFAD488A625FF896C415F5EE905B488D1E0574503C3AB499F13E2C060E69DF765A5B42FA49E1DF69F374408FAAE33A5C36C

Card certificate (Length: 190100): 308201153081BDA003020102020400000001300A06082A8648CE3D040302301431123010060355042D03090043415F526F6F7430301E170D3234303932363133353531305A170D3235303932363133353434315A30133111300F060355042D0308005375626A6563743059301306072A8648CE3D020106082A8648CE3D03010703420004D28C5A669113F1D0071F0FDE00EFFAD488A625FF896C415F5EE905B488D1E0574503C3AB499F13E2C060E69DF765A5B42FA49E1DF69F374408FAAE33A5C36CC0300A06082A8648CE3D04030203470030440220744B3D8D91FF170C485539EB26D75F9A93F640F0838EFBB41B1E9DF3827F7E2002201C613CCF5232EEF56BCFFF6ACBE102CB4E1511912E5739CDE14C460DA5D412B

CARootKey, used on card and reader side: Pub: 045662F4BB1B06816C7849CE19CAE901CC720FD300458EAD97806541280EC0F74C5D0F42692ECD2B97A93F6417D7FC93EDA6E4A38FEF81E414CCB9B87A51B72F2D; Priv: 337A05371C2B27C28267C054303264B623C4F33B0754B59655BECFC756368B4D

PICCMasterKey: AES-128, 00000000000000000000000000000000 (default value)

PICCAppDefaultKey: AES-128, 00000000000000000000000000000000 (default value)

Table 69. Creating an ECC-only application

Step	Command	Direction	Data	Comment
1	CreateApplication	>	CAA5A5A5078FEF869512C5F9AABC	Create the future ECC-only application. AID = 0xA5A5A5; Key Settings1 = 0x07, Application Master Key needs to be changeable, and the AppKeySettings needs to be frozen; KeySettings2 = 0x8F, This indicates AES-128 keys are used, and that the AppMasterTmpKey shall be enabled.
2	Success	<	004D592D622F9E9925	
3	GetApplicationIDs	>	6ABA08D0D8D24E2AEA	Get AIDs that are present on the card and select the previously generated app by using its AID.
4	Response	<	00A5A5A54E07B852009FF87A	
5	SelectApplication	>	5AA5A5A5	
6	Success	<	00	

Table 69. Creating an ECC-only application...continued

Step	Command	Direction	Data	Comment
7	AuthenticateEV2First Part1	>	710000	Symmetric authentication with the AppMaster TmpKey. This key is only there for the reason of creating the structure and keys that are used in ECC-only applications, and will later be disabled. The key value is the AppDefaultKey as configured on the card (no derivation used)
8	Response 1	<	AFDAD31010F130FE19E77C442FED65F16E	
9	AuthenticateEV2First Part2	>	AF27C0ECC2434F5832FAB227B941CC62FC438FF4CEB4DF2A011C8231FD34C21E77	
10	Response 2	<	0091DEEED446785F65E100D0BC95B4F23071CE0644397041E9DC543123578F764A	
11	ManageKeyPair	>	4600000C80003000000000E675A9F4D630752DA973A95EE354EDE21438305F74CBEBB9	The ManageKey Pair command is used to create an ECC key inside the application, that later will be used for authentication. In this case, the Option byte is set to 0x00, which means a key will be generated, and a public key will be returned. This command is sent in ComMode. Full, as it is the first creation of a key in this key entry. Later, the CommMode setting of the key itself will be used. Please note: There is no command Data in this command. However, one block (16 byte) of encrypted data are present, as it is required to pad command Data with 80 and then following 00 until the next multiple of the AES blocksize is reached.
12	Response + Public Key	<	00C0C6CB2E3F15F74F4B3A2FF32717B3F552128C9CC0E4192CE58A048E279D8E4CC85389C0FF1745E9B79EF31AC8F4C22D31A6BDED4C75EC5CBE30818EB70386A917AC893D299243D332A62896DF38817C758F7250DFCE0DF7	
13	CreateStdDataFile	>	CD000000E0000300C26DB9A4DE4A3FA4	Create a StdData File that will hold the certificate of the public key that was
14	Success	<	0078C60A065EF22534	

Table 69. Creating an ECC-only application...continued

Step	Command	Direction	Data	Comment
15	WriteData	>	3D000000001C0100190100308201153081BDA003020102020400000001300A06082A8648CE3D040302301431123010060355042D03090043415F526F	created before. Data is written in multiple frames with native chaining.
16	AdditionalFrame	<	AF	
17	WriteData	>	AF6F7430301E170D3234303932363133353531305A170D3235303932363133353434315A30133111300F060355042D0308005375626A656374305930	
18	AdditionalFrame	<	AF	
19	WriteData	>	AF1306072A8648CE3D020106082A8648CE3D03010703420004D28C5A669113F1D0071F0FDE00EFFAD488A625FF896C415F5EE905B488D1E0574503C3	
20	AdditionalFrame	<	AF	
21	WriteData	>	AFAB499F13E2C060E69DF765A5B42FA49E1DF69F374408FAAE33A5C36CC0300A06082A8648CE3D04030203470030440220744B3D8D91FF170C485539	
22	AdditionalFrame	<	AF	
23	WriteData	>	AFE826D75F9A93F640F0838EFBB41B1E9DF3827F7E2002201C613C0F5232EEF56BCFFF6ACBE102CB4E1511912E5739CDE14C460DA5D412B	
24	Success	<	00	
25	ManageCARootKey	>	48000C0F0030000000000000E199CF49700FD31479B44E7AA9D6D525222A03C8EE0D1C9A8872EAA93CE447BB44FD71309459C279131AD7567BB55E1C031266A16EFE61E1F0FB394354B0DC8DA01EFB3F6A364FB82F1781C5AEB43EAEFF0A30F66B40A7C0	This command loads a CA root key into the application, that will later be used for validating the reader certificate. With this, all requirements for an asymmetric mutual authentication are there.
26	Success	<	00DDA90EBA59E54152	

At this point, a dummy ChangeKey or ChangeKeyEV2 command can be sent, with which the temporary symmetric key will be disabled. After that, personalization can be finished by using asymmetric authentication methods.

17 Abbreviations

Table 70. Abbreviations

Abbreviation	Description
ACK	Positive ACKnowledgement
AES	Advanced Encryption Standard
AMK	Application Master Key
APDU	Application Protocol Data unit
APK	Application Key
ATS	Answer To Select
ATQA	Answer To request, Type A
C-APDU	Command APDU (sent to MIFARE DUOX)
CID	Card Identifier
CMAC	MAC according to NIST Special Publication 800-38B
CRC	Cyclic Redundancy Check
CSR	Certificate Signing Request
EDC	Error Detection Code
FDT	Frame Delay Time
FWT	Frame Waiting Time
HLTA	HALT Command, Type A
I-block	Information block
INF	Information Field
KDK	Key Derivation Key
LSB	Lowest Significant Byte
LSb	Lowest Significant bit
MAC	Message Authentication Code
NAD	Node Address
NAK	Negative AcKnowledgement
PCB	Protocol Control Byte
PCD	Proximity Coupling Device
PICC	Proximity Card
PPS	Protocol and Parameter Selection
R-APDU	Response APDU (received from MIFARE DUOX)
R-block	Receive ready block
R(ACK)	R-block containing a positive acknowledge
R(NAK)	R-block containing a negative acknowledge
RATS	Request for Answer To Select
REQA	REQuest Command, Type A

Table 70. Abbreviations...continued

Abbreviation	Description
S-block	Supervisory block
SAK	Select AcKnowledge
ShS	Shared Secret
SM	Secure Messaging
TMI	Transaction MAC Input
UID	Unique Identifier
WTX	Waiting Time extension

18 References

- [1] **Data sheet** — MIFARE DUOX contactless multi-application IC, document number 9744xx, available on <https://www.nxp.com/mynxp/secure-files>
- [2] **ISO/IEC 14443** Identification cards – Contactless integrated circuit(s) card – Proximity cards. Part 1, 2, 3, 4.
- [3] **Data sheet** – MIFARE SAM AV3 Reader Module for MIFARE product family, Document number 3235xx, available on <https://www.nxp.com/mynxp/secure-files>
- [4] **National Institute of Standards and Technology (NIST)** – Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication. NIST Special Publication 800-38B, May 2005.
- [5] **National Institute of Standards and Technology (NIST)** – Recommendation for Block Cipher Modes of Operation: Methods and Techniques. NIST Special Publication 800-38A, December 2001.
- [6] **National Institute of Standards and Technology (NIST)** – Recommendation for Key Derivation Using Pseudorandom Functions. NIST Publication 800-108, 2000.
- [7] **Data sheet** – NTAG 424 DNA - Secure NFC T4T compliant IC, document number 4654xx, available on the NXP Website: <https://www.nxp.com/docs/en/data-sheet/NT4H2421Gx.pdf>.
- [8] **Software** - CardTestFramework - tbd
- [9] **National Institute of Standards and Technology (NIST)** - Recommendation for key-derivation methods in keyestablishment schemes. NIST Special Publication 800-56C, August 2020.
- [10] **ISO/IEC 9594-8:2020 Information technology** - Open systems interconnection - Part 8: The Directory: Publickey and attribute certificate frameworks - Ninth edition, 11 2020

19 Revision history

Table 71. Revision history

Document ID	Release date	Description
AN1421 v.1.0	13 November 2024	• Initial version

Legal information

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.

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 <https://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.

Suitability for use in non-automotive qualified products — Unless this document 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.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

Translations — A non-English (translated) version of a document, including the legal information in that 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 vulnerabilities or may support established security standards or specifications with known limitations. 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.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

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

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

DESFire — is a trademark of NXP B.V.

MIFARE — is a trademark of NXP B.V.

NTAG — is a trademark of NXP B.V.

MIFARE Classic — is a trademark of NXP B.V.

Provided under NDA only
 COMPANY CONFIDENTIAL - Personal copy for:
 Sergii Salata
 Quantag IT Solutions GmbH
 Do not copy, share or reproduce
 5263157b-1e10-4fbb-8b2e-f2e3f6abec0d

Tables

Tab. 1.	ATS	6	Tab. 37.	Writing the NDEF File content	51
Tab. 2.	MIFARE DUOX ISO/IEC 7816 DF Parameters	7	Tab. 38.	Changing the file settings of the NDEF File	52
Tab. 3.	MIFARE DUOX ISO/IEC 7816 EF Parameters	8	Tab. 39.	SDM Session Key Generation	53
Tab. 4.	Accessing the EF in ISO/IEC 7816 Mode	8	Tab. 40.	Decryption of SDMEncFileData	54
Tab. 5.	ISO/IEC 14443-4 Standard Chaining in ISO/IEC 7816 Standard Framing mode, PCD uses Chaining	9	Tab. 41.	SDMMAC Validation	54
Tab. 6.	ISO/IEC 14443-4 Standard Chaining in ISO/IEC 7816 Standard Framing Mode, PICC uses Chaining	9	Tab. 42.	SDMReadCtr Retrieval using GetFileCounters command	56
Tab. 7.	MIFARE DUOX Communication Modes	11	Tab. 43.	MIFARE DUOX Supported Memory Configurations	57
Tab. 8.	Differences between Cmd.AuthenticateEV2First and Cmd.AuthenticateEV2NonFirst	14	Tab. 44.	Keys at PICC Level and their Usages	59
Tab. 9.	Example showing Cmd.AuthenticateEV2First authentication	15	Tab. 45.	Example for Cmd.InitializeKeySet in EV2 Secure Messaging	64
Tab. 10.	Asymmetric Authentication example	18	Tab. 46.	Example for Cmd.ChangeKey in EV2 Secure Messaging	65
Tab. 11.	Numerical example of session key generation during asymmetric authentication	22	Tab. 47.	Example for Cmd.FinalizeKeySet in EV2 Secure Messaging	66
Tab. 12.	Cmd.WriteData in EV2 Secure Messaging using CommMode.MAC	24	Tab. 48.	CMAC calculation 2	67
Tab. 13.	Cmd.ReadData in EV2 Secure Messaging using CommMode.MAC	25	Tab. 49.	Application Keys	68
Tab. 14.	Cmd.WriteData in EV2 Secure Messaging using CommMode.Full	26	Tab. 50.	CRL File creation	77
Tab. 15.	Cmd.ReadData in EV2 Secure Messaging using CommMode.Full	27	Tab. 51.	Create Conventional Application in EV2 Securing Messaging	80
Tab. 16.	EV2 Secure Messaging at a glance	28	Tab. 52.	Encryption of KeyID.AppDAMDefaultKey	87
Tab. 17.	Preconditions for PICCData Decryption	31	Tab. 53.	DAMMAC Generation	88
Tab. 18.	Decryption of PICCData	31	Tab. 54.	Create Delegated Application using EV2 Secure Messaging	88
Tab. 19.	SDM Session Key Generation	33	Tab. 55.	Command List associated with Access Rights	91
Tab. 20.	Create Standard Data File with SDM enabled	34	Tab. 56.	Setting of File Access Rights (mandatory Access Condition Set) during File Creation	93
Tab. 21.	Creating the NDEF application	35	Tab. 57.	Setting of File Access Rights (multiple Access Condition Sets) during File Creation	94
Tab. 22.	Selecting the NDEF application	35	Tab. 58.	Setting of Additional File Access Rights using Cmd.ChangeFileSettings	94
Tab. 23.	Authentication with AuthenticateEV2First using the Application Master key of the NDEF application	36	Tab. 59.	Command List associated to the additional SDM Access Rights	96
Tab. 24.	Creating the CC File	37	Tab. 60.	Write a Record into a LinearRecordFile	97
Tab. 25.	Creating the NDEF File	38	Tab. 61.	Create Transaction MAC File Example (in EV2 Secure Messaging)	101
Tab. 26.	Writing the CC file content	39	Tab. 62.	Example: Commit Reader ID (in EV2 Secure Messaging)	104
Tab. 27.	Changing the file settings of the CC File	39	Tab. 63.	Example: Commit Transaction (in EV2 Secure Messaging)	105
Tab. 28.	Writing the NDEF File content	41	Tab. 64.	Application creation and file creation for TSIG support	106
Tab. 29.	Changing the file settings of the NDEF File	42	Tab. 65.	Transaction with returned TSIG	107
Tab. 30.	SDM Session Key Generation	43	Tab. 66.	TSIG verification	108
Tab. 31.	SDMMAC Validation	44	Tab. 67.	Card-unilateral authentication for originality check purposes	110
Tab. 32.	Writing the NDEF File content	45	Tab. 68.	116
Tab. 33.	Changing the file settings of the NDEF File	46	Tab. 69.	Creating an ECC-only application	119
Tab. 34.	Decryption of PICCData	48	Tab. 70.	Abbreviations	122
Tab. 35.	SDM Session Key Generation	48	Tab. 71.	Revision history	125
Tab. 36.	SDMMAC Validation	48			

Figures

Fig. 1.	ISO/IEC 7816 file structure in MIFARE DUOX	7	Fig. 18.	Shared Application Indicator Usage	90
Fig. 2.	CMAC Calculation and Truncation for EV2 Secure Messaging	12	Fig. 19.	Multiple File Access Rights	92
Fig. 3.	AES Encryption in EV2 Secure Messaging	13	Fig. 20.	Coding of Access Conditions	93
Fig. 4.	AES Decryption in EV2 Secure Messaging	13	Fig. 21.	Transaction MAC Workflow using ReaderID .	100
Fig. 5.	Authentication scheme in EV2 Secure Messaging	14	Fig. 22.	Transaction MAC Computation Steps	102
Fig. 6.	Session key generation for EV2 MACing	16	Fig. 23.	MIFARE DUOX APDU embedded into T = CL Frame	111
Fig. 7.	Session key generation for EV2 Encryption	17	Fig. 24.	ISO/IEC 14443-4 Chaining on the PCD side	111
Fig. 8.	Plain communication in EV2 SM	23	Fig. 25.	ISO/IEC 14443-4 Chaining on the PICC side	112
Fig. 9.	MACed communication in EV2 SM	24	Fig. 26.	MIFARE DUOX Native C-APDU	112
Fig. 10.	Fully enciphered communication in EV2 SM	26	Fig. 27.	MIFARE DUOX Native R-APDU	112
Fig. 11.	Key Set Architecture	61	Fig. 28.	MIFARE DUOX Native C-APDU wrapped into ISO/IEC 7816 APDU format	113
Fig. 12.	Key Set Configuration after Application Creation	63	Fig. 29.	MIFARE DUOX Native R-APDU wrapped into ISO/IEC 7816 APDU format	113
Fig. 13.	Key Sets after Key Set Rolling	66	Fig. 30.	Example for reading 99 bytes in ISO/IEC 7816 wrapping (using native chaining)	114
Fig. 14.	Key Set Rolling in the Field	67	Fig. 31.	Example for writing 99 bytes in ISO/IEC 7816 wrapping (using native chaining)	115
Fig. 15.	DAM Fundamentals	82	Fig. 32.	ISO 7816-4 Standard C-APDU	117
Fig. 16.	DAM Keys Usage	83	Fig. 33.	ISO 7816-4 Standard R-APDU	117
Fig. 17.	Parameters to be agreed between PICC Owner and Application Provider	84			

Contents

1	Introduction	2	4.2.3	SDM specific Encryption of File Data	31
1.1	About this document	2	4.2.4	SDM specific SDMMAC Calculation	32
1.2	MIFARE DUOX Support Package	2	4.2.5	SDM Session Key Generation	33
1.2.1	Documents	2	4.2.6	Enabling Secure Dynamic Messaging	34
1.2.2	Software	2	4.2.6.1	Example: Enabling SDM during Standard Data File creation	34
1.2.3	Hardware	3	4.2.7	NDEF Formatting of a MIFARE DUOX application	35
1.2.4	Trainings	3	4.2.7.1	NDEF Formatting: Creating the NDEF application	35
1.3	Organization of this document	4	4.2.7.2	NDEF Formatting: Creating the CC File	36
2	ISO/IEC 14443 and MIFARE DUOX	5	4.2.7.3	NDEF Formatting: Creating the NDEF File	37
2.1	ISO/IEC 14443-3	5	4.2.7.4	NDEF Formatting: Writing the CC File content	39
2.1.1	SAK	5	4.2.7.5	NDEF Formatting: Changing the file settings of the CC File	39
2.1.2	Random ID	5	4.2.7.6	NDEF Formatting: Writing the NDEF File content and configuring it	40
2.2	ISO/IEC 14443-4	5	4.2.8	Writing the NDEF content to the NDEF application	41
2.2.1	Buffer size	5	4.2.8.1	Example: SDM Mirroring with plain PICCData	41
2.2.2	ATS	6	4.2.8.2	Example: SDM Mirroring with encrypted PICCData	45
2.2.3	ISO/IEC 14443-4 Chaining	6	4.2.8.3	Example: SDM Mirroring with encrypted File Data (SDMEncFileData)	50
3	ISO/IEC 7816 Support	7	4.2.9	SDM Counter Retrieval	56
3.1	ISO/IEC 7816 File Structure in MIFARE DUOX	7	4.2.9.1	Example: Retrieving the SDMReadCtr using GetFileCounters command	56
3.1.1	Example: ISO/IEC 7816 File Structure in MIFARE DUOX	7	5	PICC Memory and Configuration management	57
3.2	Chaining in ISO/IEC 7816-4	9	5.1	MIFARE DUOX Memory Organization	57
3.2.1	Example: PCD uses Chaining	9	5.2	Memory consumption due to Key Usage	57
3.2.2	Example: PICC uses Chaining	9	6	Symmetric Key Management	59
3.3	Security in ISO 7816-4 Standard Mode	10	6.1	Key Types	59
4	Secure Messaging	11	6.2	Key Versioning	59
4.1	EV2 Secure Messaging	12	6.3	PICC Level Keys	59
4.1.1	MAC Calculation	12	6.4	Application Level Keys	61
4.1.2	Encryption	12	6.4.1	Application Key Sets	61
4.1.3	Symmetric Authentication	14	6.4.1.1	Key Set Architecture	61
4.1.3.1	Example: Authentication using Cmd.AuthenticateEV2First	15	6.4.1.2	Key Set Personalization	62
4.1.4	Session Keys Generation	16	6.4.1.3	Initialize Key Set	63
4.1.5	Asymmetric Authentication	17	6.4.1.4	Change Keys of a Key Set	64
4.1.5.1	Example: Card personalization and mutual asymmetric authentication	18	6.4.1.5	Finalize Key Set	65
4.1.5.2	Session Keys generation for ECC authentication	22	6.4.1.6	Roll Key Set	66
4.1.6	CommMode.Plain	23	6.4.2	Application Keys	68
4.1.7	CommMode.MAC	23	7	Asymmetric Key Management	70
4.1.7.1	Example: Cmd.WriteData using CommMode.MAC in EV2 Secure Messaging	24	7.1	ManageCARootKey	70
4.1.7.2	Example: Cmd.ReadData using CommMode.MAC in EV2 Secure Messaging	25	7.2	ManageKeyPair	71
4.1.8	CommMode.Full	25	8	Certificate Management	72
4.1.8.1	Example: Cmd.WriteData using CommMode.Full in EV2 Secure Messaging	26	8.1	Card Certificates	72
4.1.8.2	Example: Cmd.ReadData using CommMode.Full in EV2 Secure Messaging	27	8.2	Reader Certificates	72
4.1.9	EV2 Secure Messaging Summary	28	8.3	Certificate Generation	74
4.2	Secure Dynamic Messaging	29	8.3.1	OpenSSL for creating certificates	74
4.2.1	SDM for Reading from a File	29	8.4	Certificate Revocation	76
4.2.2	SDM specific Encryption of PICCData	31	9	Application Management	79
4.2.2.1	Example: Decryption of PICCData	31			

9.1	Conventional Application Management	79	12.1.3.1	Example: Create Transaction MAC File in EV2 Secure Messaging	101
9.1.1	Example: Create Conventional Application in EV2 Secure Messaging	79	12.1.4	Transaction MAC Computation Steps	102
9.1.2	Delete Conventional Application	80	12.1.4.1	Transaction MAC Computation – Prestep	102
9.2	Delegated Application Management (DAM) - using symmetric keys	81	12.1.4.2	Transaction MAC Computation – Initialization	102
9.2.1	Steps needed for enabling Delegated Application Management	81	12.1.4.3	Transaction MAC Computation – Update	103
9.2.2	Responsibilities of the card issuer	81	12.1.4.4	Transaction MAC Computation – CommitReaderID	104
9.2.3	DAM Fundamentals	81	12.1.4.5	Transaction MAC Computation – Finalization	105
9.2.4	DAM Step 1: Personalization of the Card by the Card Issuer	82	12.1.5	Retrieval of TMC and TMV	105
9.2.4.1	DAM Keys	82	12.2	Transaction SIG	105
9.2.4.2	Enabling the DAM keys	83	13	Proximity Check	109
9.2.5	DAM Step 2: Agreement between the PICC Owner and Application Provider (On the DAM Parameters)	84	14	Originality Check	110
9.2.5.1	DAMMAC Generation	85	15	MIFARE DUOX APDUs	111
9.2.5.2	Encryption of the KeyID.AppDAMDefaultkey	85	15.1	MIFARE DUOX Command – Response APDU (Framing)	111
9.2.6	DAM Step 3: Application Provider creates its Application	85	15.2	MIFARE DUOX Native APDUs	112
9.2.7	Important DAM Parameters	85	15.3	ISO/IEC 7816 wrapping of MIFARE DUOX Native APDUs	113
9.2.7.1	DAM Slot Number	85	15.3.1	Example: Cmd.ReadData using ISO/IEC 7816 APDU Wrapping	114
9.2.7.2	DAM Slot Version	86	15.3.2	Example: Cmd.WriteData using ISO/IEC 7816 APDU Wrapping	114
9.2.7.3	Quota Limit	86	15.4	Example: Cmd.WriteData – Writing to FileType.SdtDataFile using ISO/IEC 7816 Wrapping and CMAC communication	116
9.2.7.4	Application DAM Default Key	86	15.5	ISO/IEC 7816 Standard APDUs	117
9.2.8	Example: Create Delegated Application	87	16	Additional Hints and Recommendations ...	118
9.2.8.1	Create Delegated Application Example (in EV2 SM)	88	16.1	Key Management	118
9.2.9	Delete Delegated Application	89	16.2	Application Management	118
9.2.10	Format Delegated Application	89	16.3	Initial trust of an ECC only application	118
9.3	Shared Application	90	17	Abbreviations	122
10	File Management	91	18	References	124
10.1	File Access Rights Management	91	19	Revision history	125
10.1.1	Enabling of File Access Conditions	92		Legal information	126
10.1.1.1	Different Kinds of Access Conditions	93			
10.1.1.2	Example: Setting of File Access Rights (mandatory Access Condition Set) during File Creation	93			
10.1.1.3	Example: Setting of File Access Rights (multiple Access Condition Sets) during File Creation	94			
10.1.2	Changing of File Access Conditions	94			
10.1.2.1	Example: Setting of Additional Access Rights by using Cmd.ChangeFileSettings	94			
10.1.2.2	Example: Multiple Access Condition Scenario	95			
10.1.3	File Access Conditions related to SDM	95			
11	Data Management	97			
11.1	Example: Using the Linear Record File	97			
12	Transaction Management	99			
12.1	Transaction MAC	99			
12.1.1	Committing the Reader ID	100			
12.1.2	Transaction MAC Enabling	100			
12.1.3	Create Transaction MAC File Command	101			

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