AN12343

MIFARE DESFire Light Features and Hints

Rev. 1.1 — 20 January 2020 Application note
522511 COMPANY PUBLIC

Document information

Information	Content
Keywords	MIFARE, MIFARE DESFire Light, MIFARE DESFire EV2, ISO/IEC 7816, Secure Messaging, LRP, Leakage Resilient Primitive.
Abstract	This application note addresses the usage of MIFARE DESFire Light, explains single commands in detail and illustrates sequential transactions by displaying the full APDU exchange. Additionally it gives hints for usage of the IC and the command implementation.

Revision history

Revision history

Rev	Date	Description
1.1	20200120	Added chapter Section 11, Originality Checking
1.0	20190131	Initial version of the document

1 Abbreviations

Table 1. Abbreviations

Acronym	Description
AID	Application Identifier
APDU	Application Protocol Data Unit
ATQA	Answer to Request-A, according to ISO/IEC 14443
ATS	Answer to Select, according to ISO/IEC 14443
C-APDU	Command-APDU
CTR	Counter
DF Name	Dedicated File Name. The DF Name to be used for the ISO Select command as defined in ISO 7816-4. According to EMV, the DF Name is also called AID (application identifier). To avoid confusion with the MIFARE DESFire Application ID (AID) this document uses the term DF Name for the ISO Select command.
DIV	Diversification
ISO Select	The ISO Select command used with the DF Name (ISO Select by DF Name), including the FCI response. Definition according to ISO/IEC 7816-4.
LRICB	Leakage Resilient Indexed Codebook
LRP	Leakage Resilient Protocol
NFC	Near Field Communication
PCD	Proximity Coupling Device (reader, terminal)
PDCap	Physical Device Capabilities
PICC	Proximity Integrated Circuit Card
PRNG	Pseudo Random Number Generator
PTO	Public Transport Operator
R-APDU	Response-APDU
SAK	Select Acknowledge
TLV	Tag-Length-Value formatting
UID	Unique IDentifier

2 Introduction

MIFARE DESFire Light is the latest member of the MIFARE DESFire family. The MIFARE DESFire family offers products which are based on a flexible, secure and scalable platform, offering continuous innovation and the important aspects security as well as privacy. As contrast to MIFARE DESFire EV2 which is a very powerful, flexible and complex platform for dynamic multi-application use cases, the MIFARE DESFire Light is a simple and secure platform tailor-made for single application use cases.

The key aspects of MIFARE DESFire Light are simplicity, scalability as well as security and privacy, offering the perfect solution for single application usage scenarios.

In this document, the features and functionality of MIFARE DESFire Light is presented and explained in detail. Useful implementation hints are given, in order to make the usage and integration of MIFARE DESFire Light into new or already existing infrastructures as easy as possible.

Many commands are illustrated in detail and also cryptographic functionalities are discussed with the help of examples in order to facilitate the implementation. It is recommended to use a Secure Access Module (SAM) or Secure Key storage Element (SE) inside a MIFARE DESFire Light reading device, to establish an even higher level of overall system security.

2.1 About the content of this document

This document addresses developers, project leaders and system integrators who have a general technical understanding or are already familiar with the MIFARE DESFire product family. Knowledge of the reader terminal infrastructure or complete service infrastructure is good to have.

Note that this document does not cover the general working principle of the MIFARE DESFire Light. Read Ref [3]. in order to get the full overview and description of MIFARE DESFire Light and the associated command set.

In case you want to compare the MIFARE DESFire Light functionality with the MIFARE DESFire EV2, refer to following data sheet, Ref [4].

This application note is a supplementary document for implementations using the MIFARE DESFire Light. Should there be any confusion, check the MIFARE DESFire Light data sheet Ref [3]. The best use of this application note is achieved by reading the mentioned data sheet in advance.

2.2 Structure of this document

This document describes the relevant information for designing and implementing a MIFARE DESFire Light system, starting with an Introduction.

Section 3 explains the details of the ISO/IEC 7816 compatibility of the MIFARE DESFire Light IC.

Following this, Section 4 goes into details of available options for memory and configuration management of the chip.

Afterwards, Section 5 elaborates on the pre-configured MIFARE DESFire Light application and the different application selection methods.

Section 6 then goes into detail of file management and how access rights can be associated to them.

The next section, namely Section 7, describes the available secure messaging mechanisms as well as the associated MACing and encryption algorithms in detail.

How exchanging data with the IC works, is outlined in Section 8.

A closer look on available keys, especially application keys and how they can be used and changed, is given in Section 9.

Concluding the technical details, Section 10 discusses the details of the underlying transaction management of the MIFARE DESFire Light IC.

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

Table 2. Annotation of Symbols

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

Note: All given numerical sample transactions are just examples, describing the usage of commands and providing reference values to verify an implementation. If the command parameters consist of multiple bytes, they are represented in the LSB (least significant byte) first format in the given examples. Furthermore, any data, value and cryptogram is expressed in hexadecimal format if not otherwise specified.

2.3 MIFARE DESFire Light Support Package

Like all other MIFARE products, MIFARE DESFire Light comes 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:

Documents:

1. **Data sheet – MIFARE DESFire Light**
   Product Data sheet, available in NXP DocStore (doc.no. 4307xx*[1] and on the NXP website under the following weblink:https://www.nxp.com/docs/en/data-sheet/MF2DL_H_x0.pdf
2. **Application note – AN12341 MIFARE DESFire Light Quick Start Guide**
   available in NXP DocStore, document number 5224xx and on the NXP website under the following weblink: https://www.nxp.com/docs/en/application-note/AN12341.pdf
3. **Application note – AN12343 MIFARE DESFire Light Features and Hints**
   available in NXP DocStore, document number 5225xx and on the NXP website under the following weblink: https://www.nxp.com/docs/en/application-note/AN12343.pdf
4. **Application note – AN12342 MIFARE DESFire Light Card Coil Design Guide**
   available in NXP DocStore, document number 5226xx and on the NXP website under the following weblink: https://www.nxp.com/docs/en/application-note/AN12342.pdf
5. **Application note – AN12344 MIFARE DESFire Light Target Applications and Usage**
   will be available in NXP DocStore, document number 5227xx
6. **Product Qualification Package – PQP5235 MIFARE DESFire Light**

1 xx … document version number

available in NXP DocStore, document number 5235xx

7. **Wafer Specification – WS4475 MIFARE DESFire Light Wafer Specification** available in NXP DocStore, document number 4475xx
8. **Application note – AN12304 Leakage Resilient Primitive (LRP) Specification** available in NXP DocStore, document number 4660xx and on the NXP website under the following weblink: https://www.nxp.com/docs/en/application-note/AN12304.pdf

Software:

- RFID Discover – a software tool to evaluate MIFARE DESFire EV2 and send commands to the card, can be ordered via your NXP contact. It comes in two different versions:
  - SW2144 RFIDDiscover Lite – Contains the full implementation and functionality for MIFARE DESFire Light, but only a reduced, very limited command set of MIFARE DESFire EV1 and EV2. It can be downloaded publicly on the NXP website.
  - SW1866xx RFIDDiscover – Contains additionally to the MIFARE DESFire Light support also the full command set of MIFARE DESFire EV1 and EV2 and should be used to make use of the full card functionality of these products. It can be obtained via NXP DocStore.

Hardware:

MIFARE DESFire Light samples can be ordered via several different channels

- The NXP eSample desk
- Your NXP contact person
- Your preferred NXP reseller or distribution partner

Trainings:

MIFARE DESFire Light 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: https://www.nxp.com/support/training-events/nxp-technology-days:NXP-TECH-DAYS.

2.4 MIFARE DESFire Light Product Compatibility

MIFARE DESFire Light is functional compatible with MIFARE DESFire EV2, and systems which are running on MIFARE DESFire Light can support MIFARE DESFire EV2 out of the box.

MIFARE DESFire Light supports a selected set of commands which are available also on MIFARE DESFire EV2. These selected commands are the essential ones which are needed for an easy-to-use single application card, providing a maximum of convenience and simplicity.

For more advanced infrastructure designs, an up-scale to additional MIFARE DESFire EV2 commands can happen in parallel. This can be interesting for systems which also want to support multi-application scenarios and implement more advanced features that only MIFARE DESFire EV2 offers.

MIFARE DESFire Light applications can be issued with exactly the same structure and content also on MIFARE DESFire EV2 ICs, which ensures terminal interoperability and easy portable applications. Furthermore, MIFARE DESFire Light applications can

be made available via the NXP AppXplorer, a collaboration platform for card issuers and service providers. This makes it possible to advertise a MIFARE DESFire Light application and provide it via the NXP AppXplorer to a very wide range of customers. From there, it can be downloaded easily and installed over-the-air on any MIFARE DESFire EV2 card in the market.

2.5 MIFARE DESFire Product Family

MIFARE DESFire is a product family, not a single product.

The first member of the MIFARE DESFire family was the MIFARE DESFire D40 (or also called MIFARE DESFire EV0), which was released in 2002 and discontinued in 2011. The second member is MIFARE DESFire EV1, which was released in 2008 with three different memory versions. The third evolution is MIFARE DESFire EV2, which was released in 2016.

The latest add-on to the MIFARE DESFire family is the MIFARE DEFire Light, offering a subset of MIFARE DESFire EV2 commands and being fully compatible with MIFARE DESFire EV2.

Furthermore, in the family, there are the following main product groups existing:

- MIFARE DESFire native standalone ICs
- MIFARE DESFire implementations on microcontrollers
- MIFARE DESFire JCOP applets (applets on Java card)
- MIFARE DESFire HCE solution as MIFARE 2GO

To distinguish between the broad variety of MIFARE DESFire products at a reader terminal, the MIFARE DESFire specific GetVersion command can be used. This command returns necessary information of the product, which can be used for an accurate product type identification.

Most likely byte 2 of frame 1, of the GetVersion response, which gives information related to the used main hardware product type identifier, is sufficient to categorize the device which was presented to the reader. Multiple possibilities of the setting of byte 2 are given in Table 3.

Table 3. MIFARE DESFire Product Types and byte 2 (HW product type) of the response of GetVersion command

Product Type	Byte 2
MIFARE DESFire native IC (physical card)	0x01
MIFARE DESFire Light native IC (physical card)	**0x08**
MIFARE DESFire implementation on microcontroller (physical card)	0x81, 0x83
MIFARE DESFire applet on Java card / secure element	0x91
MIFARE DESFire HCE (MIFARE 2GO)	0xA1

3 ISO/IEC 7816 Support

MIFARE DESFire Light supports some Inter Industry Instructions as described in ISO/IEC 7816-4, to comply with several application standards, e.g. APTA, RIS. The full set of supported instructions is described in [3]

3.1 ISO/IEC 7816 File Structure in MIFARE DESFire Light

In MIFARE DESFire Light, the application and file structure is pre-defined and fully compliant to the ISO/IEC 7816 suggested file system layout.

According to the ISO/IEC 7816 standard, the PICC level is called Master File (MF), applications are called Dedicated File (DF) and files are named Elementary Files (EF). The standard ISO/IEC 7816 file system layout is displayed in Figure 1.

To uniquely identify an application, it is characterized by a so-called DF Name. Additionally, an ISO File ID can be assigned to the application. On file level, the identification happens via the ISO File ID only.

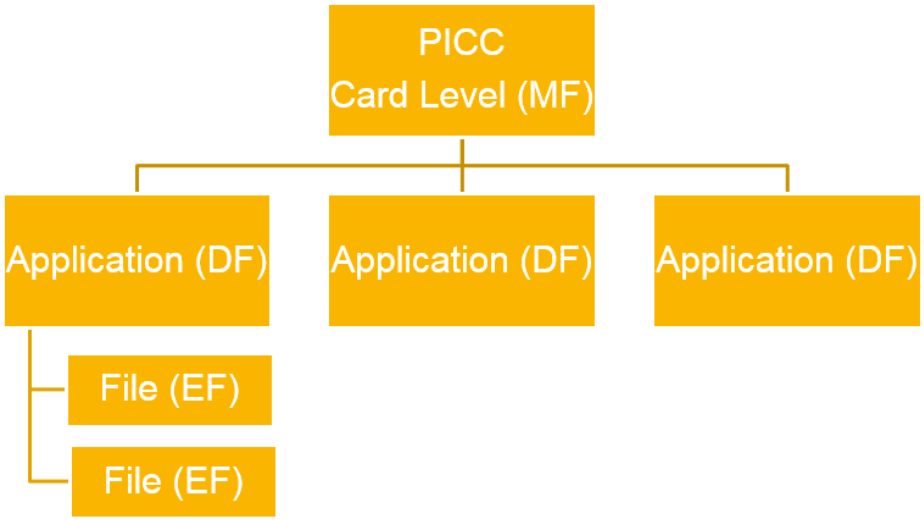


Figure 1. ISO/IEC 7816 file system structure

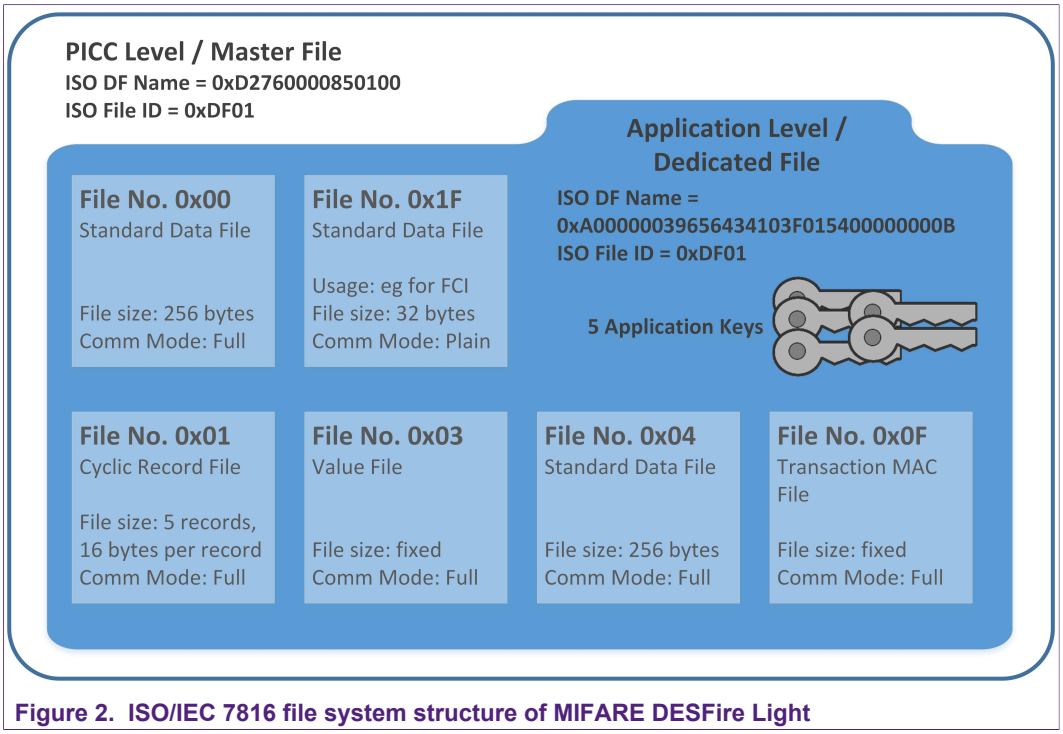


Figure 2. ISO/IEC 7816 file system structure of MIFARE DESFire Light

In MIFARE DESFire Light this ISO/IEC 7816 file structure is configured in a fixed way, with only one available application (Dedicated File) which contains a limited number of files (Elementary Files).

4 Memory and Configuration Management

Commands of this section can be used to read-out and to configure settings concerning the full IC or to modify application attributes.

4.1 Example: Executing GetVersion

The GetVersion command can be used to retrieve manufacturing related data of the IC. The command is executed in free frames, and data that is returned includes information about the hardware, software and production-related data concerning the IC.

In example Table 4, the GetVersion command is executed without being authenticated directly after selecting the MIFARE DESFire Light IC and specific application.

Table 4. Executing Cmd.GetVersion for retrieving manufacturing related data of the PICC

Step	Command		Data Message
1	**Select the IC by sending Cmd.ISOSelect**		
2	Cmd.ISOSelect C-APDU (Cmd || Ins || P1 || P2 || Lc || Data || Le)	>	00A4040C10A000000396564341 03F015400000000B00
3	Cmd.ISOSelect R-APDU (SW1 || SW2)	<	9000 (SUCCESS)
	Constructing the full GetVersion Command APDU		
4	CLA	=	90
5	Ins	=	60
6	P1	=	00
7	P2	=	00
8	Lc (Length of the data)	=	- (not present, as no command data is present)
9	Data	=	- (no command data is present for this command)
10	Le (Length expected)	=	00
11	Cmd.GetVersion C-APDU (Part 1) (Cmd || Ins || P1 || P2 || Lc || Data || Le)	>	9060000000
12	Cmd.GetVersion R-APDU (Part 1) (Response Data || SW1 || SW2)	<	0408013000130591AF
13	R-APDU contained SW1 || SW2	=	91AF with 0xAF meaning the request for one additional frame, in this case for GetVersion Part 2
14	Ins	=	AF (additional frame)
15	Cmd.GetVersion C-APDU (Part 2)	>	90AF000000
16	Cmd.GetVersion R-APDU (Part 2) (Response Data || SW1 || SW2)	<	04080130000A000408010000130591AF
17	R-APDU contained SW1 || SW2	=	91AF with 0xAF meaning the request for one additional frame, in this case for GetVersion Part 2

Step	Command		Data Message
18	Ins	=	AF (additional frame)
19	Cmd.GetVersion C-APDU (Part 3)	>	90AF000000
20	Cmd.GetVersion R-APDU (Part 3) (Response Data || SW1 || SW2)	<	04080130001305040801000 00B0004DE5F1EACC040FFFFFFFFFFF40179100
21	R-APDU contained SW1 || SW2	=	9100 (SUCCESS)

4.2 Example: Executing SetConfiguration

The SetConfiguration command can be used to configure card or application-related attributes.

In example Table 5, the SetConfiguration command is used, to modify the parameters of the pre-installed value file inside the MIFARE DESFire Light application.

The modification which is executed, is setting the upper limit of the value file to 1000 (0x03E8) and setting the option "Free GetValue" to disabled, meaning an authentication is forced for retrieving the value.

Table 5. Executing Cmd.SetConfiguration in CommMode.Full and Option 0x09 for updating the Value file configuration

Step	Command		Data Message
1	Option of the SetConfiguration command (modify the Value File configuration)	=	09
2	Data (FileNo || Lower Limit || Upper Limit || Value || ValueOption)	=	0300000000E80300000000000001
3	FileNo	=	03
4	Lower Limit	=	00000000
5	Upper Limit	=	E8030000
6	Value	=	00000000
7	ValueOption	=	01 (meaning Free GetValue not allowed, LimitedCredit enabled)
8	TI	=	55B1B324
9	CmdCounter	=	0200
10	SesAuthENCKey	=	B2FF2B22D5E2695A24A8A3ECB9C4AD8F
11	SesAuthMACKey	=	D7C0206A531AC9E03560F31CA5BD5179
	Encrypting the Command Data		
12	IV	=	00000000000000000000000000000000
13	IV_Input (IV_Label || TI || CmdCounter || Padding)	=	A55A55B1B32402000000000000000000
14	IV for CmdData = Enc(KSesAuthENC, IV_Input)	=	17665980F48A3AA90281A28B8500F011

Step	Command		Data Message
15	Data (Data || Padding)	=	0300000000E8030000000000000018000
16	Encrypted Data = E(KSesAuthENC, Data)	=	AB30CF86A5E36F372FFAEA00FE25264E
17			
18	**Generating the MAC for the Command APDU**		
19	IV	=	00000000000000000000000000000000
20	MAC_Input (Ins || CmdCounter || TI || CmdHeader || Encrypted Data)	=	5C020055B1B32409AB30CF86A5E36F372FFAEA00FE25264E
21	MAC = CMAC(KSesAuthMAC, MAC_Input)	=	AEBDF6A497B94265
	Constructing the full SetConfiguration Command APDU		
22	CLA	=	90
23	Ins	=	5C
24	P1	=	00
25	P2	=	00
26	Lc (Length of the data)	=	19
27	Data (CmdHeader || Encrypted Data || MAC)	=	09AB30CF86A5E36F372FFAEA00FE25264EAEBDF6A497B94265
28	Le (Length expected)	=	00
	Exchanging the SetConfiguration Command and Response APDU		
29	Cmd.SetConfiguration C-APDU (Cmd || Ins || P1 || P2 || Lc || Data || Le)	>	905C00001909AB30CF86A5E36F372FFAEA00FE25264EAEBDF6A497B9426500
30	CmdCounter	=	0300
31	R-APDU	<	E30A7457B6CBEAFF9100
32	Response Code (RC)	=	9100 (00 = SUCCESS)
33	Received MAC	=	E30A7457B6CBEAFF
	Verifying the received Response MAC		
34	MAC_Input (RC || CmdCounter || TI)	=	00030055B1B324
35	MAC = CMAC(KSesAuthMAC, MAC_Input)	=	E30A7457B6CBEAFF

In example Table 6, the SetConfiguration command is used, to modify the ATS of the chip.

Table 6. Executing Cmd.SetConfiguration in CommMode.Full and Option 0x02 for updating the ATS

Step	Command		Data Message
1	Option of the SetConfiguration command (modify the Value File configuration)	=	02
2	Data (ATS = TL ||| T0 || TA || TB || TC || Tk)	=	067777710280
3	TI	=	1096DAF6
4	CmdCounter	=	0000
5	SesAuthENCKey	=	FBEF52B7E739C525403B6769973A0147
6	SesAuthMACKey	=	E32E831F49AB417A530E2A65256AEB68
	Encrypting the Command Data		
7	IV	=	00000000000000000000000000000000
8	IV_Input (IV_Label || TI || CmdCounter || Padding)	=	A55A1096DAF600000000000000000000
9	IV for CmdData = Enc(KSesAuthENC, IV_Input)	=	19533040E805DF2C9594B52C2FE7C879
10	Data (Data || Padding)	=	06777771028080000000000000000000
11	Encrypted Data = E(KSesAuthENC, Data)	=	B61698C5E6597DBFE452B606E8E65943
	Generating the MAC for the Command APDU		
12	IV	=	00000000000000000000000000000000
13	MAC_Input (Ins || CmdCounter || TI || CmdHeader || Encrypted Data)	=	5C00001096DAF602B61698C5E6597DBFE452B606E8E65943
14	MAC = CMAC(KSesAuthMAC, MAC_Input)	=	CBA381A2FD372405
	Constructing the full SetConfiguration Command APDU		
15	CLA	=	90
16	Ins	=	5C
17	P1	=	00
18	P2	=	00
19	Lc (Length of the data)	=	19

Step	Command		Data Message
20	Data (CmdHeader || Encrypted Data || MAC)	=	02B61698C5E6597DBFE452B606E8E65943CBA381A2FD372405
21	Le (Length expected)	=	00
	Exchanging the SetConfiguration Command and Response APDU		
22	Cmd.SetConfiguration C-APDU (Cmd || Ins || P1 || P2 || Lc || Data || Le)	>	905C00001902B61698C5E6597DBFE452B606E8E65943CBA381A2FD37240500
23	CmdCounter	=	0100
24	R-APDU	<	277990D1805C9ADD9100
25	Response Code (RC)	=	9100 (00 = SUCCESS)
26	Received MAC	=	277990D1805C9ADD
	Verifying the received Response MAC		
27	MAC_Input (RC || CmdCounter || TI)	=	0001001096DAF6
28	MAC = CMAC(KSesAuthMAC, MAC_Input)	=	277990D1805C9ADD

4.3 Example: Retrieving the UID from the card by using GetCardUID

The UID of the IC is typically returned during the card activation process. However, if there is the need to retrieve the unique ID of the chip in a different way, the GetCardUID command can be used for retrieving it.

In example Table 7, the GetCardUID command is executed, after selecting the MIFARE DESFire Light application and authenticating with Cmd.AuthenticateEV2First. This means, that the GetCardUID command is executed using AES secure messaging.

Table 7. Executing Cmd.GetCardUID for retrieving the 7 byte UID of the card

Step	Command		Data Message
1	**Select the IC by sending Cmd.ISOSelect**		
2	Cmd.ISOSelect C-APDU (Cmd || Ins || P1 || P2 || Lc || Data || Le)	>	00A4040C10A000000396564341 03F015400000000B00
3	Cmd.ISOSelect R-APDU (SW1 || SW2)	<	9000 (SUCCESS)
	Authenticating using AES secure messaging		
4	KeyNo	=	00
5	KeyValue	=	01234567890123456789012345678901

Step	Command		Data Message
6	Cmd.AuthenticateEV2First C-APDU (Part 1)	>	9071000002000000
7	R-APDU (Part 1) (E(Kx, RndB)) || SW1 || SW2 Response Code SW2 = 0xAF = Additional Frame	<	B9FC6CCAE153125C7C17E6906433C0F491AF
8	Cmd.AuthenticateEV2First C-APDU (Part 2)	>	90AF000020C8B3AFDEC10EE8298471A7B41736B4381BA1BE0F57F66387C5577721B70F847F00
9	R-APDU (Part 2) E(Kx, TI || RndA' || PDcap2 || PCDcap2) || Response Code Response Code = 0x9100 = SUCCESS	<	8138FD2450891FCDB4935D9F19C30B55FAD52DC54086933E0FBEC3DE9266BD809100
10	TI	=	569D4B24
11	CmdCounter (is reset to 0000 after a successful Cmd.AuthenticateEV2First command)	=	0000
12	SesAuthENCKey	=	C1D7BD9F60034D8432F9AF3403D573D0
13	SesAuthMACKey	=	FD9E26C9766F07C1D07106C0F8F3671F
	Constructing the full GetCardUID Command APDU		
14	CLA	=	90
15	Ins	=	51
16	P1	=	00
17	P2	=	00
18	Lc (Length of the data)	=	08
19	Data = MAC over command = $MAC_{KSesAuthMACKey}$(Ins || CmdCtr || TI)	=	$MAC_{KSesAuthMACKey}$(510000569D4B24) = ED5CB7A932EF8D7C2E91B42A1139F11B Truncated MAC = every 2nd uneven byte of the MAC = 5CA9EF7C912A391B
20	Le (Length expected)	=	00
21	Cmd.GetCardUID C-APDU (Part 1) (Cmd || Ins || P1 || P2 || Lc || Data || Le)	>	90510000085CA9EF7C912A391B00
22	Cmd.GetCardUID R-APDU (Part 1) ($E_{KSesAuthENC}$(Response Data) || Response MAC || SW1 || SW2)	<	CDFFBF6D34231DA2789DA9D3AB15D560CE75E39EDBE94C2F9100
23	R-APDU contained SW1 || SW2	=	9100 (SUCCESS)
24	$E_{KSesAuthENC}$(Response Data)	=	CDFFBF6D34231DA2789DA9D3AB15D560
25	Response MAC	=	CE75E39EDBE94C2F

Step	Command		Data Message
	Decrypting the received Response Data		
26	Starting IV	=	00000000000000000000000000000000
27	IV_Input_Response = 0x5A || 0xA5 || TI || CmdCtr || 0x0000000000000000	=	5AA5569D4B2401000000000000000000
28	IV_Response for RespData = $E_{KSesAuthENC}$(IV_Input)	=	5A42ECB2111A9267FA5F2682523229AC
29	IV_Response	=	5A42ECB2111A9267FA5F2682523229AC
30	Decrypting the Response Data = $D_{KSesAuthENC}$($E_{KSesAuthENC}$(Response Data))	=	$D_{KSesAuthENC}$(CDFFBF6D34231DA2789DA9D3AB15D560) = 04DE5F1EACC040800000000000000000
31	Decrypted Response Data = (UID || Padding)	=	04DE5F1EACC040800000000000000000
32	UID	=	04DE5F1EACC040
	Verifying the received Response MAC		
33	Calculated Response MAC = $MAC_{KSesAuthMACKey}$(RC || CmdCtr + 1 || TI || Encrypted Response Data)	=	$MAC_{KSesAuthMACKey}$(000100569D4B24CDFFBF6D34231DA2789DA9D3AB15D560) = Truncated MAC = every 2nd uneven byte of the MAC = CE75E39EDBE94C2F
34	Response MAC	=	CE75E39EDBE94C2F

5 Application Management

In contrast to the other MIFARE DESFire family members, MIFARE DESFire Light comes already with one pre-configured application. No new applications can be created, but only the existing one can be used. The number of available applications on MIFARE DESFire Light therefore is fixed to one.

Inside the pre-configured application, there are already files pre-created as well, which can be used for data operations. The file structure is depicted in Figure 2.

After selecting the application, data which is stored inside the files of the application can be accessed and updated. Therefore an authentication might be necessary, based on the access rights of the individual files.

5.1 Application Selection

The pre-configured MIFARE DESFire Light application needs to be selected by using the standard inter-industry ISO/IEC 7816 command called ISOSelectFile. For this application selection, one of the application identifiers needs to be supplied, namely either the ISO File ID or the ISO DF Name.

An example, on how application selection is working based on the ISO DF Name is given in Table 8.

Furthermore another example, on how application selection is working based on the ISO File ID is given in Table 9.

5.1.1 Application Selection by using ISOSelectFile and the ISO DF Name

Table 8. Application Selection by using ISOSelectFile and the ISO DFName

Step	Command		Data Message
1	ISO DFName	=	A000000396564341 03F015400000000B
2	Cmd	=	00
3	Ins	=	A4
4	P1	=	04
5	P2	=	0C
6	Lc (Length of the data)	=	10
7	Data	=	A000000396564341 03F015400000000B
8	Le (Length expected)	=	00
9	Cmd.ISOSelect C-APDU (ISOSelect Application by DFName)	>	00A4040C10A000000396564341 03F015400000000B00
10	R-APDU	<	9000 (SUCCESS)

5.1.2 Application Selection by using ISOSelectFile and the ISO File ID

Table 9. Application Selection by using ISOSelectFile and the ISO FileID

Step	Command		Data Message
1	ISO FileID	=	DF01
2	Cmd	=	00

Step	Command		Data Message
3	Ins	=	A4
4	P1	=	00
5	P2	=	00
6	Lc (Length of the data)	=	02
7	Data	=	DF01
8	Le (Length expected)	=	00
9	Cmd.ISOSelect C-APDU (ISOSelect Application by FileID)	>	00A4000002DF0100
10	R-APDU	<	9000 (SUCCESS)

6 File Management

MIFARE DESFire Light maintains user data in form of files inside the application. Upon IC delivery, six files of the following different file types are already pre-installed in the application:

- Standard Data File
- Cyclic Record File
- Value File
- Transaction MAC File (not intended to store user data, but needed for the TMAC feature)

A file can be identified uniquely inside the application based on the File Number or ISO/IEC 7816-4 File ID. The file numbers of all available files can be retrieved from the application with the GetFileIDs command, as explained in Section 6.2. Furthermore the specific configuration settings of a file can be retrieved from the IC by using the GetFileSettings command.

As the MIFARE DESFire Light IC comes with a defined application and file structure, it is not intended to delete or resize the files. Of the above mentioned files, only the Transaction MAC File can be deleted. This is needed to reconfigure the Transaction MAC key, which is vital for calculating the TMAC value.

Information is stored within the files in different ways: as raw data (bytewise), as value or in the form of formatted records. Depending on the need, the appropriate files can be chosen for data storage. Files can be accessed in different manner, e.g. in form of reading, writing or updating data.

Furthermore, access to the files can be restricted with a file access rights management, see Section 6.1.

To guarantee data consistency, all files except the Standard Data File, implement a backup mechanism. This backup mechanism follows a transaction-oriented approach by mirroring the user data in a shadow image. 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, after the complete transaction is finished. The transaction mechanism is described in more detail in Section 10.

6.1 File Access Rights Management

File data is accessed with three different access rights Read, Write and ReadWrite. Each of these access rights is permitting the use of a subset of commands, see the detailed list in Table 10.

In addition, an access right called Change is specified per file, permitting Cmd.ChangeFileSettings to change the file access rights and the communication mode of the file. An access right is granted if one condition associated to it is satisfied. A condition for granting access is either the authentication with the related key of the access right, or the access right is set to the value 0xE.

Table 10. Command list associated with file access rights

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

Each file is associated with one set of access conditions. Each set contains the four access conditions: Read, Write, ReadWrite, Change.

Important Note: The access rights management of the Transaction MAC file is slightly different to the general access rights management of the other file types. For this special file type, the access right Write is always set to 0xF (never), so writing is never possible to this file, and the access right ReadWrite is set to the key number which is needed to commit the reader ID to the card (the key which is needed for authentication before sending the Cmd.CommitReaderID), or to 0xF in case the reader ID is not mandatorily to be committed. Read and Change access rights are treated in the same way as for all other file types.

The file access conditions are already defined for the files in the MIFARE DESFire Light application. They can be adapted or changed later on using the Cmd.ChangeFileSettings. An example for changing the file access rights can be seen in Table 11.

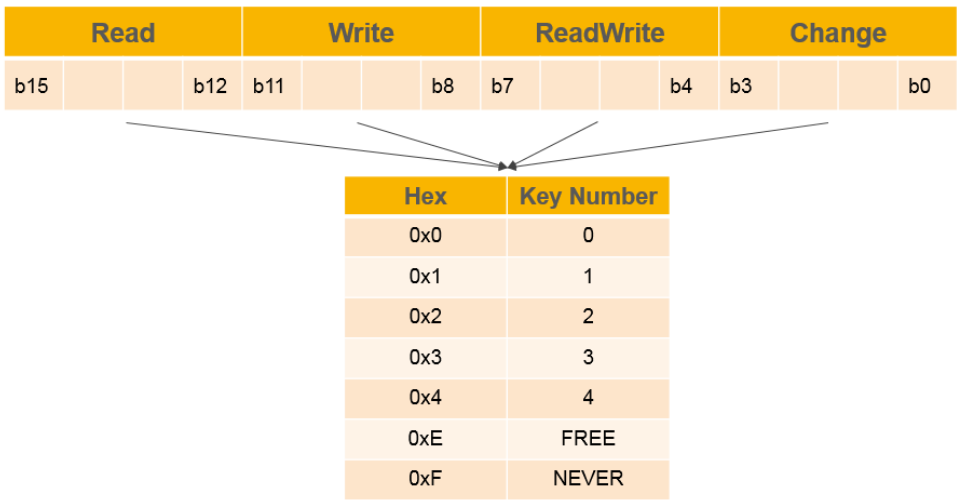
6.1.1 Different kinds of Access Conditions

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 Application Key is needed (key numbers 0x0 – 0x4)
- Free access (0xE)
- No access / Access forbidden (0xF)

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 3.



Hex	Key Number
0x0	0
0x1	1
0x2	2
0x3	3
0x4	4
0xE	FREE
0xF	NEVER

Figure 3. Coding of File Access Conditions

For MIFARE DESFire Light, the file access rights have been defined as depicted in Figure 4. They can be reconfigured by using the command Cmd.ChangeFileSettings.

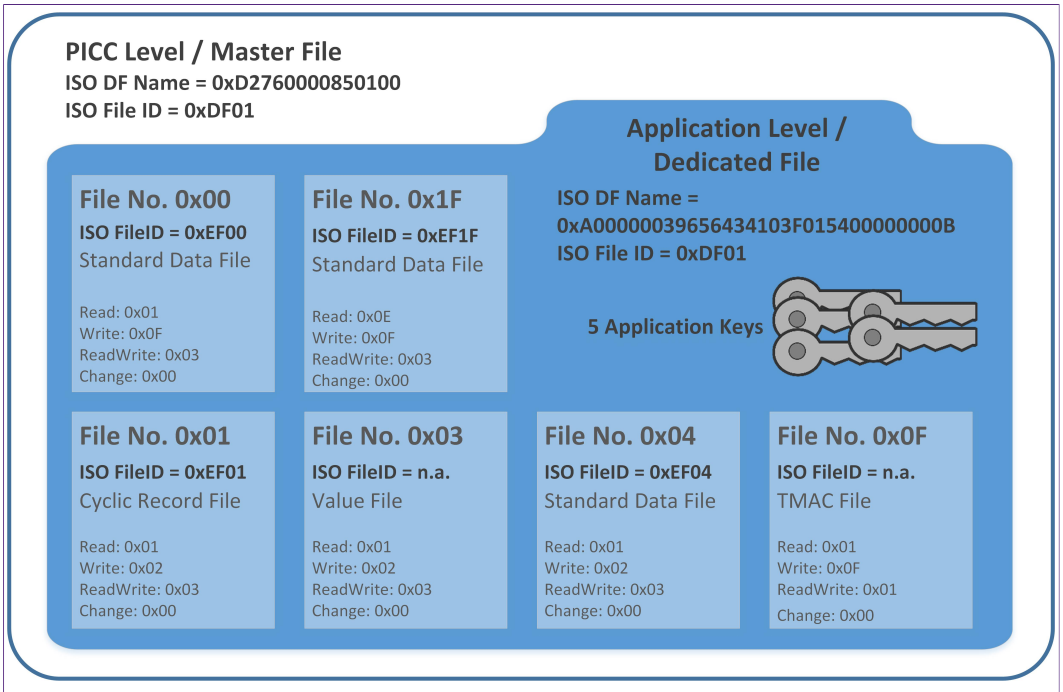


Figure 4. MIFARE DESFire Light File Access Rights

6.1.2 Change the File Access Rights by using ChangeFileSettings

The Cmd.ChangeFileSettings can be used to modify the overall settings of a file. The file settings which can be changed are the file access rights, the communication mode and the TMC Limit.

This specific example of Cmd.ChangeFileSettings changes the file access rights from the default values (which are 0x1230 for the access conditions Read-Write-ReadWrite-Change) to the values 0xEEEE.

The command is executed in AES Secure Messaging and uses the FULL communication mode. As a pre-step the application was selected and Cmd.AuthenticateEV2First with the key that was specified in the previous Change access right was executed.

Table 11. ChangeFileSettings in AES Secure Messaging

Step	Command		Data Message
1	ISO DFName	=	EF04
2	FileNo	=	04
3	FileOption	=	00
4	TMCLimit	=	00000000
5	TMCLimitLenght	=	04
6	AccessRights	=	EEEE (ReadWrite-Change-Read-Write)
7	CommMode	=	03 (Fully encrypted)
8	IV_Label	=	A55A
9	Transaction Identifier (TI)	=	6E0F8127
10	CmdCounter	=	0000
	Encrypting the Command Data		
11	IV_Input (IV_Label || TI || CmdCounter || Padding)	=	A55A6E0F81270000000000000000000000
12	IV	=	00000000000000000000000000000000
13	KSesAuthENC	=	7535C7D52756FB31F8D6AD4CCC64395A
14	IV for CmdData = Enc(KSesAuthENC, IV_Input)	=	CCBE102D72C1518AF2B4A05EAF9E87A2
15	Data (FileOption || AccessRights || Padding)	=	00EEEE8000000000000000000000000000
16	Encrypted Data = E(KSesAuthENC, Data Input)	=	A7AA7229D8260CE6802F5125C4DF4DDB
	Generating the MAC for the Command APDU		
17	IV	=	00000000000000000000000000000000
18	KSesAuthMAC	=	D21AC82602C130832E36262B18808FB0
19	MAC_Input (Ins || CmdCounter || TI || CmdHeader || Encrypted Data)	=	5F00006E0F812704A7AA7229D8260CE6802F5125C4DF4DDB
20	MAC = CMAC(KSesAuthMAC, MAC_Input)	=	AF84B43856AD04B6
	Constructing the full Command APDU		

Step	Command		Data Message
21	Cmd	=	90
22	Ins	=	5F
23	P1	=	00
24	P2	=	00
25	Lc (Length of the data)	=	19
26	Data (CmdHeader || Encrypted Data || MAC)	=	A7AA7229D8260CE6802F5125C4DF4DDBAF84B43856AD04B6
27	Le (Length expected)	=	00
	Exchanging the Command and Response APDU		
28	Cmd.ChangeFileSettings C-APDU (Cmd || Ins || P1 || P2 || Lc || Data || Le)	>	005F00001904A7AA7229D8260CE6802F5125C4DF4DDBAF84B43856AD04B600
29	CmdCounter	=	0100
30	R-APDU	<	64B39EF781568B789100
31	Response Code (RC)	=	9100 (00 = SUCCESS)
32	MAC_Input (RC || CmdCounter || TI)	=	0001006E0F8127
33	MAC = CMAC(KSesAuthMAC, MAC_Input)	=	64B39EF781568B78

6.2 Example: Retrieving File IDs and File Settings from the IC

In this example, the GetFileIDs command is used for retrieving all available files (identified by the File ID) from the MIFARE DESFire Light application. The command does not require any authentication and can be executed directly after selecting the application.

Furthermore, after retrieving the File ID list, one of the files is being examined further and the settings of this file are requested by using the GetFileSettings command.

Table 12. Executing Cmd.GetFileIDs and GetFileSettings to retrieve file related information from the IC

Step	Command		Data Message
	Select the IC by sending Cmd.ISOSelect		
1	Cmd.ISOSelect C-APDU (Cmd || Ins || P1 || P2 || Lc || Data || Le)	>	00A4040C10A000000396564341 03F015400000000B00
2	Cmd.ISOSelect R-APDU (SW1 || SW2)	<	9000 (SUCCESS)
	Constructing the full GetFileIDs Command APDU		

Step	Command		Data Message
3	CLA	=	90
4	Ins	=	6F
5	P1	=	00
6	P2	=	00
7	Lc (Length of the data)	=	- (NA, as no data is transferred for this command)
8	Data	=	-
9	Le (Length expected)	=	00
10	Cmd.GetFileIDs C-APDU (Part 1) (Cmd || Ins || P1 || P2 || Le)	>	906F000000
11	Cmd.GetFileIDs R-APDU (Part 1) (Response Data || SW1 || SW2)	<	0F1F030001049100
12	R-APDU contained SW1 || SW2	=	9100 (SUCCESS)
13	Response Data (six available File IDs on the IC)	=	0x0F, 0x1F, 0x03, 0x00, 0x01, 0x04
	Constructing the full GetFileSettings Command APDU for FileNo 0x00		
14	FileNo	=	00
15	CLA	=	90
16	Ins	=	F5
17	P1	=	00
18	P2	=	00
19	Lc (Length of the data)	=	01
20	Data (FileNo)	=	00
21	Le (Length expected)	=	00
22	Cmd.GetFileSettings C-APDU (Part 1) (Cmd || Ins || P1 || P2 || Lc || Data || Le)	>	90F50000010000
23	Cmd.GetFileSettings R-APDU (Part 1) (Response Data || SW1 || SW2)	<	0003301F0001009100
24	R-APDU contained SW1 || SW2	=	9100 (SUCCESS)
25	Response Data (FileType || FileOption || AccessRights || FileSize)	=	0003301F000100
26	FileType	=	00 = Standard Data File
27	FileOption (CommMode)	=	03 = Fully Encrypted
28	AccessRights	=	301F (Read = 0x1, Write = 0xF, ReadWrite = 0x3, Change = 0x0)
29	FileSize	=	000100 (256 bytes)

Step	Command		Data Message
	Constructing the full GetFileSettings Command APDU for FileNo 0x03		
30	FileNo	=	03
31	CLA	=	90
32	Ins	=	F5
33	P1	=	00
34	P2	=	00
35	Lc (Length of the data)	=	01
36	Data (FileNo)	=	03
37	Le (Length expected)	=	00
38	Cmd.GetFileSettings C-APDU (Part 1) (Cmd || Ins || P1 || P2 || Lc || Data || Le)	>	90F50000010300
39	Cmd.GetFileSettings R-APDU (Part 1) (Response Data || SW1 || SW2)	<	02033012000000000FFFFFF7F00000000039100
40	R-APDU contained SW1 || SW2	=	9100 (SUCCESS)
41	Response Data (FileType || FileOption || AccessRights || FileSize)	=	02033012000000000FFFFFF7F0000000003
42	FileType	=	02 = Value File
43	FileOption (CommMode)	=	03 = Fully Encrypted
44	AccessRights	=	3012 (Read = 0x1, Write = 0x2, ReadWrite = 0x3, Change = 0x0)
45	LowerLimit	=	00000000
46	UpperLimit	=	FFFFFF7F
47	LimitedCreditValue	=	00000000
48	LimitedCreditEnabled	=	03 (Free access to Cmd.GetValue and Cmd.LimitedCredit is enabled)

In this example, the GetISOFileIDs command is used for retrieving all available files (identified by the ISO 7816-4 File ID) from the MIFARE DESFire Light application. The command does not require any authentication and can be executed directly after selecting the application. The ISO File IDs are an alternative to the native File IDs, also uniquely identifying the files inside the application.

Table 13. Executing Cmd.GetISOFileIDs to retrieve a list of ISO 7816-4 compliant File IDs from the IC

Step	Command		Data Message
	Select the IC by sending Cmd.ISOSelect		

Step	Command		Data Message
1	Cmd.ISOSelect C-APDU (Cmd || Ins || P1 || P2 || Lc || Data || Le)	>	00A4040C10A000000396564341 03F015400000000B00
2	Cmd.ISOSelect R-APDU (SW1 || SW2)	<	9000 (SUCCESS)
	Constructing the full GetISOFileIDs Command APDU		
3	CLA	=	90
4	Ins	=	61
5	P1	=	00
6	P2	=	00
7	Lc (Length of the data)	=	- (NA, as no data is transferred for this command)
8	Data	=	-
9	Le (Length expected)	=	00
10	Cmd.GetISOFileIDs C-APDU (Cmd || Ins || P1 || P2 || Le)	>	9061000000
11	Cmd.GetISOFileIDs R-APDU (Response Data || SW1 || SW2)	<	1FEF00EF01EF04EF9100
12	R-APDU contained SW1 || SW2	=	9100 (SUCCESS)
13	Response Data (four available ISO 7816-4 File IDs on the IC)	=	0x1FEF, 0x00EF, 0x01EF, 0x04EF

7 Secure Messaging

MIFARE DESFire Light supports two secure messaging modes:

- AES Secure Messaging
- LRP Secure Messaging

Based on the chosen authentication command, one of the two secure messaging modes gets activated.

The only key type that it supported by MIFARE DESFire Light is AES, meaning the key type AES can be used for both secure messaging modes.

After an authentication between PCD and MIFARE DESFire Light PICC is performed successfully, the following conditions become true:

- A trusted and secured communication channel is established
- The access rights related to the authentication key are enabled
- Two session keys are derived from the original authentication key (both in AES and LRP secure messaging)

Important 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 ongoing authenticated session, the IV is recalculated for each command that is exchanged.
- 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 ongoing authenticated session, the IV is recalculated for each command that is exchanged.
- Any error reported by the PICC invalidates the current session of 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 DESFire Light supports three different communication modes as shown in Table 14.

Table 14. MIFARE DESFire Light Communication Modes

Communication Mode	Bit Representation	Description
Plain	x0	Plain data exchange. No encryption or MACing is used at all.
MAC	01	MACed data exchange. The data is transferred in plain, but a 8 byte long CMAC is added to the APDU to guarantee data integrity and authenticity.

Communication Mode	Bit Representation	Description
Full	11	Fully encrypted data exchange. The data is transferred encrypted and additionally secured with a 8 byte long CMAC. This is the most secure option and guarantees full data confidentiality, integrity and authenticity.

Important Note:

- If there is no active authentication, both command and response are sent in plain (command will be rejected in case an authentication is required).
- Authentication commands as well as some other commands have their own secure messaging rules and define exceptions.

7.1 AES Secure Messaging

The AES Secure Messaging gets activated after successful authentication with Cmd.AuthenticationEV2First (command code 0x71) or Cmd.AuthenticateEV2NonFirst (command code 0x77). The only supported key type AES. AES Secure Messaging is supported by MIFARE DESFire Light, and works in the same way as EV2 Secure Messaging which is supported by MIFARE DESFire EV2.

When being not authenticated (in state VCState.DFNotAuthenticated) a first authentication needs to be established, using Cmd.AuthenticateEV2First. If an authentication was already executed using Cmd.AuthenticateEV2First, and AES Secure Messaging is already established, a subsequent authentication can be achieved by executing Cmd.AuthenticateEV2NonFirst.

Cmd.AuthenticateEV2NonFirst is intended for any subsequent authentication after Cmd.AuthenticateEV2First in a transaction. Details on when to use which authentication can be seen in Table 15.

7.1.1 MAC Calculation

MACs which are used for AES Secure Messaging are calculated using the underlying AES block cipher according to CMAC standard described in [7]. The output of the 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 5.

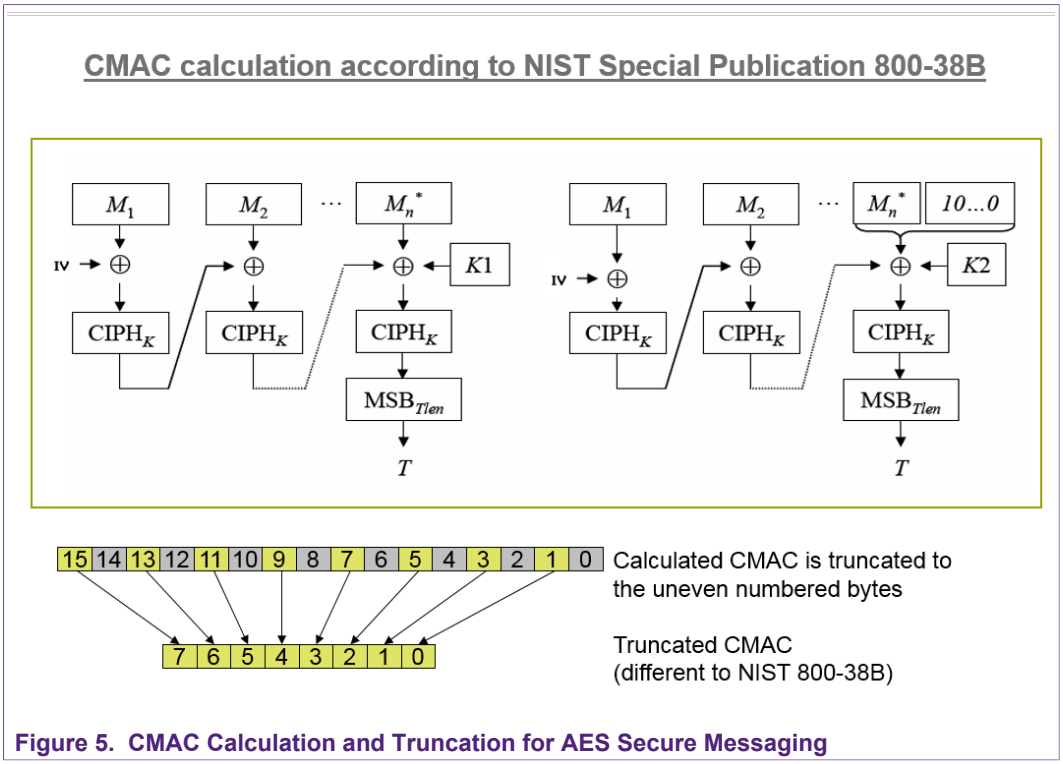


Figure 5. CMAC Calculation and Truncation for AES Secure Messaging

7.1.2 Encryption and Decryption

Encryption and decryption are done using the underlying block cipher (in this case the AES block cipher) according to the CBC mode of the NIST Special Publication 800-38A, see [6]. 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 only exception is during the authentication itself, where no padding is applied at all.

The initialization vector (IV) which is used for encrypting the data which is sent between the PCD and the PICC is always constructed by encrypting the following pre-defined set of values with the KeyID.SesAuthENCKey. The encryption algorithm used is the AES block cipher according to the ECB mode of NIST SP800-38A [6].

The pre-defined pattern for the IV calculation is the following:

- IV for CmdData = E(KSesAuthENC, 0xA5 || 0x5A || TI || CmdCtr || 0x0000000000000000)
- IV for RespData = E(KSesAuthENC, 0x5A || 0xA5 || TI || CmdCtr || 0x0000000000000000)

The CmdCtr to be used for the IV calculation always represents the current value. Details see in Section 8.1.2 in [3]. For the encryption during the authentication using Cmd.AuthenticateEV2First and Cmd.AuthenticateEV2NonFirst, the IV is always a zero-bytes IV with a length of 16 bytes.

Below images, Figure 6 and Figure 7, depict the encryption and decryption operations.

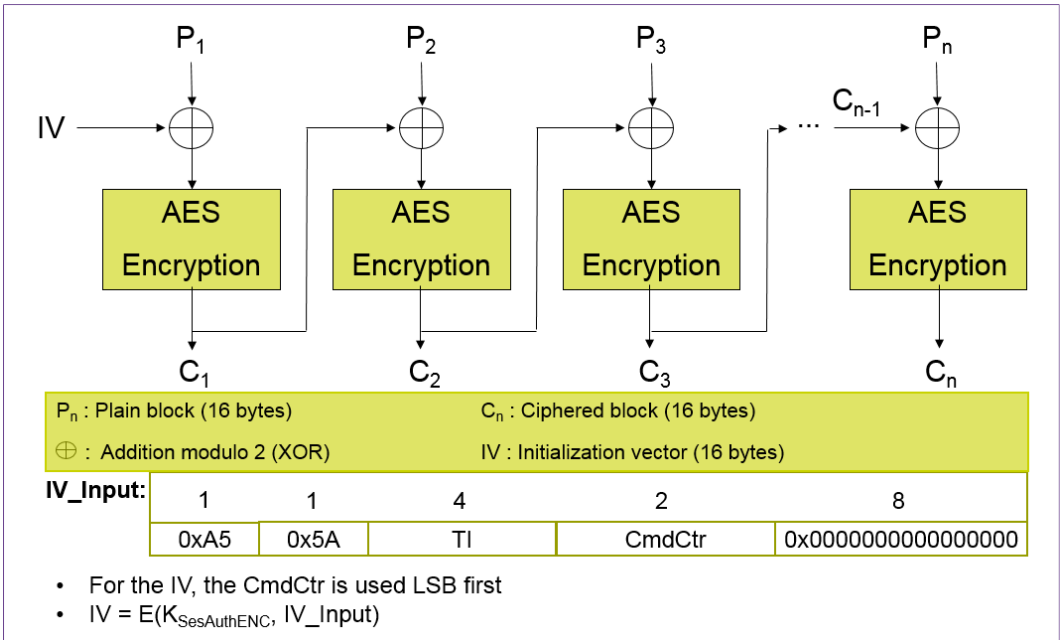


Figure 6. AES Encryption in AES Secure Messaging

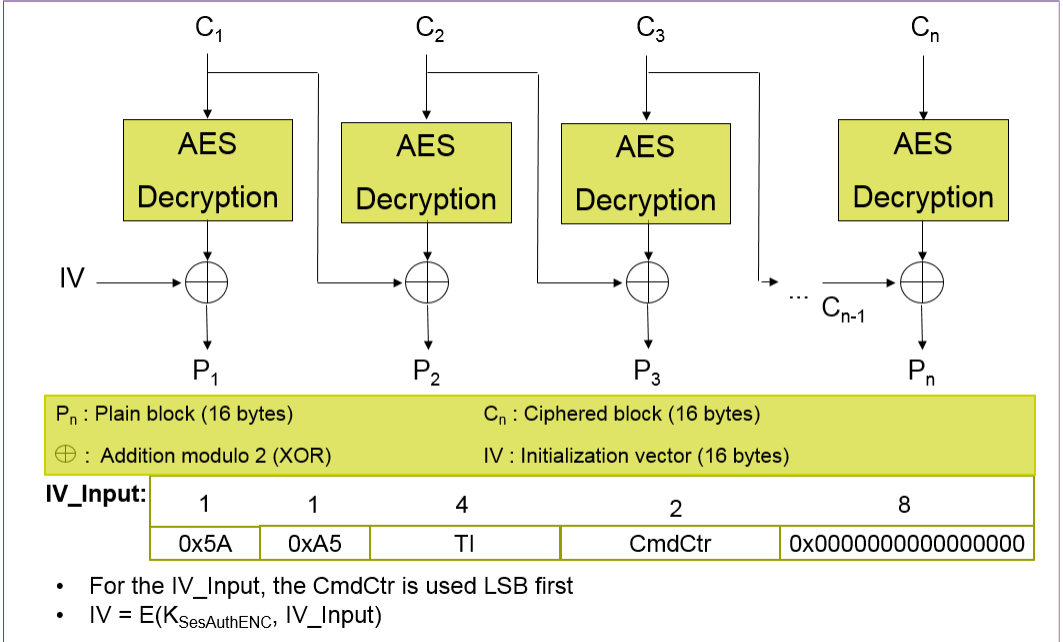


Figure 7. AES Decryption in AES Secure Messaging

7.1.3 Authentication

For AES Secure Messaging, two authentication commands Cmd.AuthenticateEV2First and Cmd.AuthenticateEV2NonFirst are available. The following table, Table 15, shows the differences between the two authentication commands.

Table 15. Differences between the commands AuthenticateEV2First and AuthenticateEV2NonFirst in AES Secure Messaging

	Cmd.AuthenticateEV2First	Cmd.AuthenticateEV2NonFirst
Purpose	To start a new transaction	To start a new session within the ongoing transaction
Cmd Code	0x71	0x77
Capability Bytes	PCD and PICC capability bytes are exchanged (PDcap2, PCDcap2)	No capability bytes are exchanged
Transaction Identifier	A new transaction identifier is generated which remains valid for the full transaction	No new transaction identifier is generated (old one remains and is reused)
Command Counter	CmdCtr is reset to 0x0000	CmdCounter stays active and continues counting from the current value
Session Keys	New session keys are 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 fewer bytes are exchanged. Any of these two authentication methods can always be used for an authentication using the KeyID.OriginalityKeys. However, 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 8

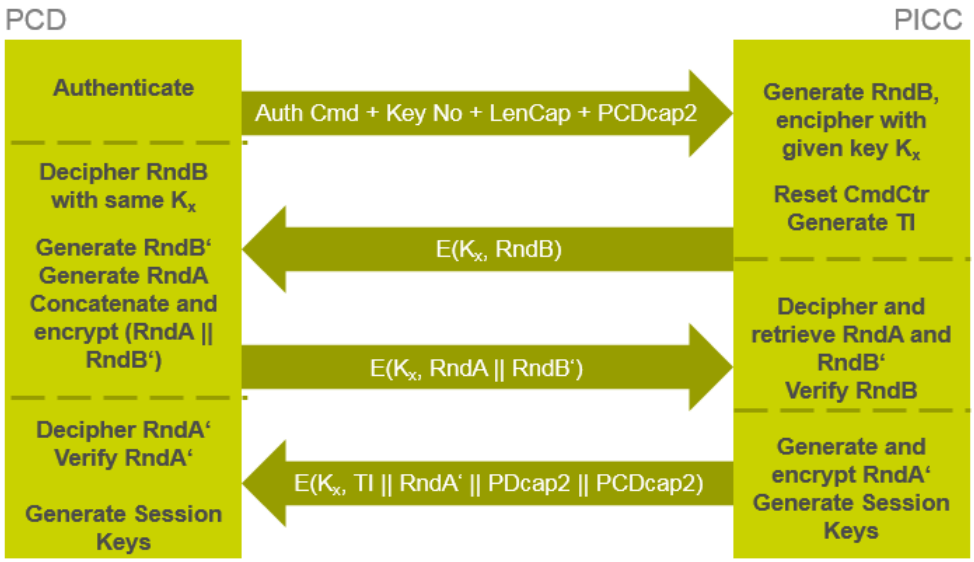


Figure 8. 3-Pass mutual authentication in AES Secure Messaging

7.1.3.1 Example: Authentication using AuthenticateEV2First

In this example, an authentication using the Cmd.AuthenticateEV2First is executed, establishing successfully the AES secure messaging.

The key number which is used for the authentication is key 0x00, the application master key, with the key default value.

Key Number = 0x00

Key Value = 0x00000000000000000000000000000000

Table 16. Authentication using Cmd.AuthenticateEV2First

Step	Command		Data Message
1	KeyNo	=	00
2	KeyValue	=	00000000000000000000000000000000
3	Cmd	=	90
4	Ins	=	71
5	P1	=	00
6	P2	=	00
7	Lc (Length of the data)	=	02
8	Data	=	0000
9	Le (Length expected)	=	00
10	Cmd.ISOAuthenticateEV2First C-APDU (Part 1)	>	9071000002000000
11	R-APDU (Part 1) (E(Kx, RndB)) || SW1 || SW2 Response Code SW2 = 0xAF = Additional Frame	<	24677DDBD46349E623798FD729006E7991AF
12	D(Kx, RndB) IV = 00000000000000000000000000000000	=	FA659AD0DCA738DD65DC7DC38612AD81
13	RndB'	=	659AD0DCA738DD65DC7DC38612AD81FA
14	RndA	=	B04D0787C93EE0CC8CACC8E86F16C6FE
15	RndA'	=	4D0787C93EE0CC8CACC8E86F16C6FEB0
16	(E(Kx, RndA || RndB'))	=	3B50445F21D21D77D500794DEB245E5A754F5F901844259F4C9B31A5C7335ACD
17	Ins	=	AF
18	Data = (E(Kx, RndA || RndB'))	=	3B50445F21D21D77D500794DEB245E5A754F5F901844259F4C9B31A5C7335ACD
19	Cmd.AuthenticateEV2First C-APDU (Part 2)	>	90AF0000203B50445F21D21D77D500794DEB245E5A754F5F901844259F4C9B31A5C7335ACD00

Step	Command		Data Message
20	R-APDU (Part 2) E(Kx, TI || RndA' || PDcap2 || PCDcap2) || Response Code Response Code = 0x9100 = SUCCESS	<	04C6DBD67417ED0D31DDDE4D2E3FFAC2B4B074F638EEF7FFF9254963B65C77599100
21	D(Kx, TI || RndA' || PDcap2 || PCDcap2) IV = 00000000000000000000000000000000	=	8CF141F34D0787C93EE0CC8CACC8E86F16C6FEB0000000000000000000000000
22	TI	=	8CF141F3
23	RndA'	=	4D0787C93EE0CC8CACC8E86F16C6FEB0
24	PDcap2	=	000000000000
25	PCDcap2	=	000000000000
26	PCD compares send and received RndA	=	B04D0787C93EE0CC8CACC8E86F16C6FE =? B04D0787C93EE0CC8CACC8E86F16C6FE
27	CmdCounter (is reset to 0000 after a successful Cmd.AuthenticateEV2First command)	=	0000
28	SesAuthENCKey	=	63DC07286289A7A6C0334CA31C314A04
29	SesAuthMACKey	=	774F26743ECE6AF5033B6AE8522946F6

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 7.1.4. In this example, the session keys have the following values:

KeyID.SesAuthENCKey = 0x63DC07286289A7A6C0334CA31C314A04

KeyID.SesAuthMACKey = 0x774F26743ECE6AF5033B6AE8522946F6

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

7.1.4 Session Key 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, [8], in counter mode. The used key derivation function is the CMAC algorithm according to NISP SP 800-38B, [7].

Two session keys are generated:

- SesAuthMACKey for MACing of APDUs, shown in Figure 9
- SesAuthENCKey for encryption and decryption of APDUs, shown in Figure 10

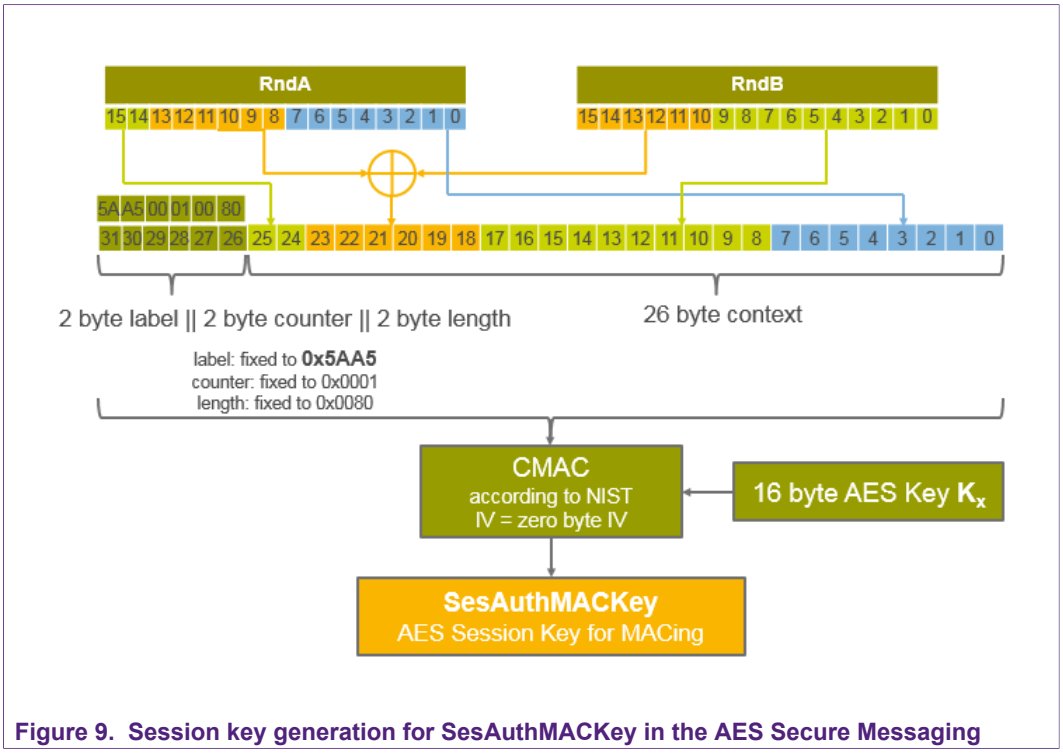


Figure 9. Session key generation for SesAuthMACKey in the AES Secure Messaging

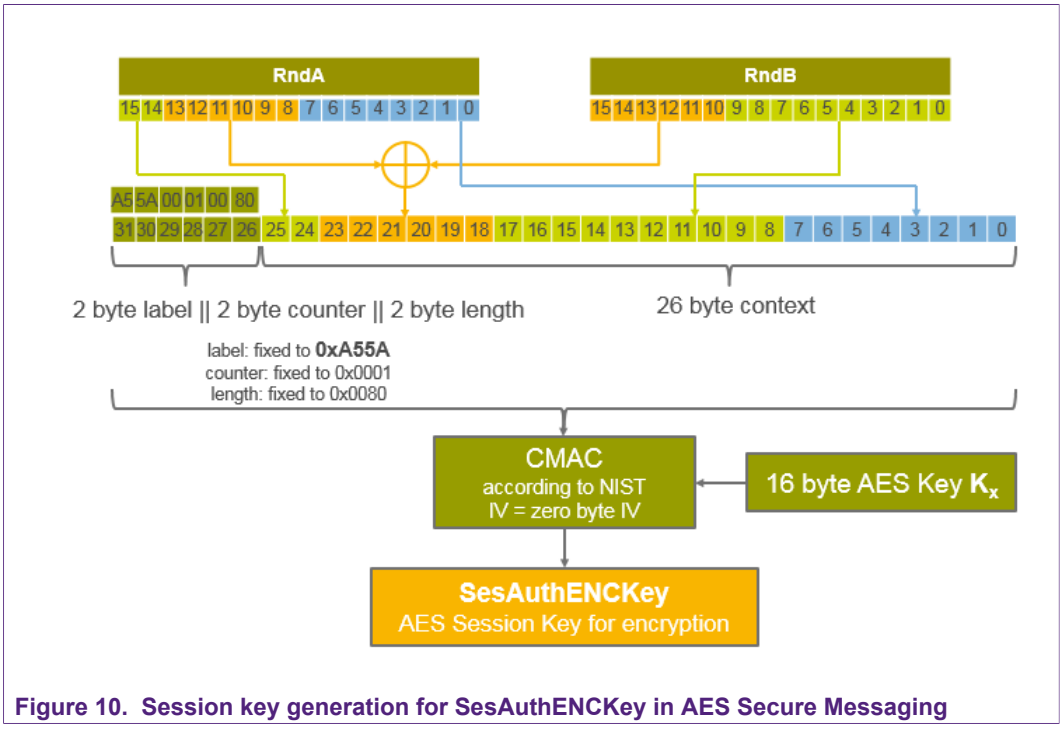


Figure 10. Session key generation for SesAuthENCKey in AES Secure Messaging

7.1.5 Plain Communication Mode

In this communication mode, the data is sent in plain between PCD and PICC, as depicted in Figure 11. The command counter is incremented also in this communication mode, so that any subsequent command that is sent in MAC communication mode or

Full communication mode can detect illegal insertion or deletion of commands. The command counter increase happens between sending the command and response APDU. More details to the command counter can be found in section 8.1.2 in [3].

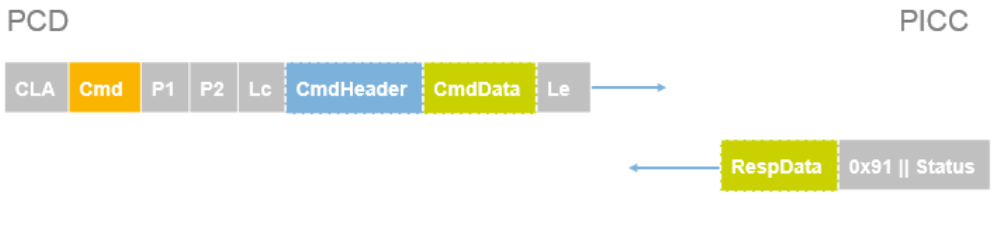


Figure 11. Plain communication mode in AES Secure Messaging

7.1.6 MAC Communication Mode

In case the MAC communication mode is chosen for file operations or is required by certain commands, the MAC is calculated by using the SesAuthMACKey of the currently ongoing session. The actual MAC calculation is described in Section 7.1.1.

For the C-APDU, the MAC is calculated over the command code, 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).

The MAC is then appended to the C-APDU respective to the R-APDU, as visible in Figure 12.

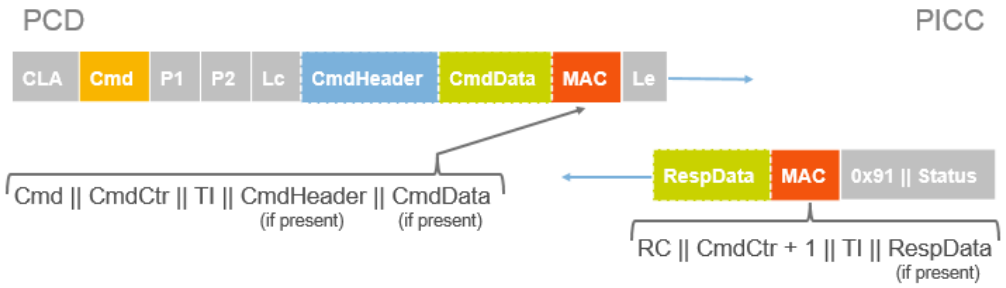


Figure 12. MAC communication mode in AES Secure Messaging

7.1.7 Full Communication Mode

If the full encrypted communication mode (Full) is chosen for data exchange, then the command data is encrypted with the SesAuthEncKey and additionally a MAC is generated using the SesAuthMACKey. The session keys are taken from the currently ongoing session.

Data is transferred fully encrypted and additionally the APDU is secured by a MAC.

The MAC is calculated as described in Section 7.1.6, and the encryption which is applied is described in Section 7.1.2.

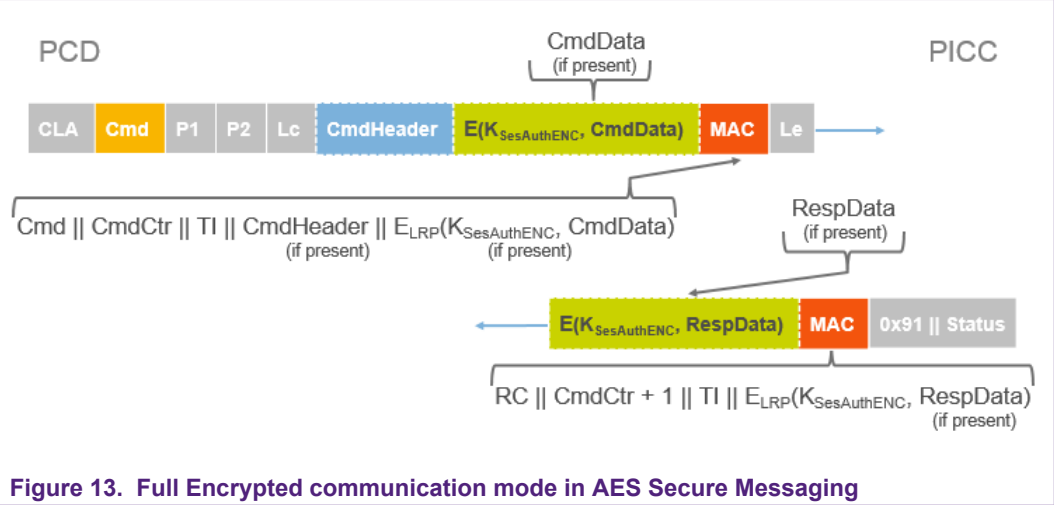


Figure 13. Full Encrypted communication mode in AES Secure Messaging

7.2 LRP Secure Messaging

The LRP Secure Messaging gets activated after successful authentication with Cmd.AuthenticateLRPFirst (command code 0x71) or Cmd.AuthenticateLRPNonFirst (command code 0x77). The only supported key type is AES. LRP Secure Messaging is supported by MIFARE DESFire Light, and adds a Leakage Resilient Primitive wrapper around AES-128.

The LRP Secure Messaging can be enabled permanently with Cmd.SetConfiguration. An example how exactly this can be done is given in Section 7.2.5.2. As the LRP Secure Messaging authentication commands are using the same command code as the authentication commands of the AES Secure Messaging, the distinguation between both happens via the parameter PCDCap2.1. For AES Secure Messaging this parameter needs to be set to 0x00, whereas for LRP Secure Messaging it needs to be set to 0x02.

When using LRP Secure Messaging and being not authenticated (in state VCState.DFNotAuthenticated) a first authentication needs to be established, using Cmd.AuthenticateLRPFirst. If an authentication was already executed using Cmd.AuthenticateLRPFirst, and LRP Secure Messaging is already established, a subsequent authentication can be achieved by executing Cmd.AuthenticateLRPNonFirst. Cmd.AuthenticateLRPNonFirst is intended for any subsequent authentication after Cmd.AuthenticateLRPFirst in a transaction. Details on when to use which authentication can be seen in Table 17.

7.2.1 MAC Calculation

MACs which are used for LRP Secure Messaging are calculated using the AES block cipher according to the LRP CMAC calculation procedure described in [9]. The output of the described LRP-CMAC calculation is a 16 byte MAC, which is further adopted for LRP 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 14.

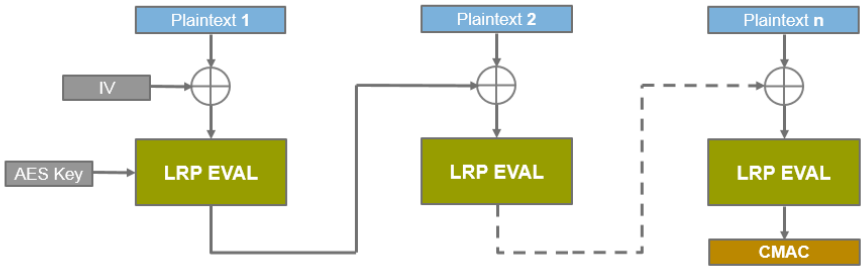


Figure 14. LRP-specific MACing (CMAC-LRP) in LRP Secure Messaging

7.2.2 Encryption and Decryption

The LRP encryption algorithm is called LRICB and operates on the structure of the AES block cipher in CTR mode. The counter that is used for the CTR mode is increased by one for each 16 byte long plaintext input block that is processed.

The core of the LRICB algorithm is the LRP EVAL function. The output of the LRP EVAL function is taken as an input for the LRICB function, as a key to encrypt 16 byte long plaintext input blocks by using the AES block cipher in EBC mode. The output of this encryption is a 16 byte long ciphertext block.

Details to the LRP encryption and decryption functionality as well as to the LRICB and LRP EVAL algorithms can be found in [9].

Below images, Figure 15 and Figure 16, depict the encryption and decryption operations.

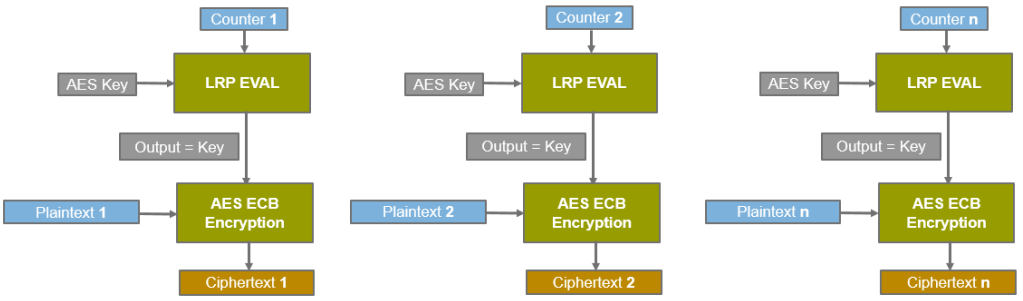


Figure 15. LRP Encryption in LRP Secure Messaging

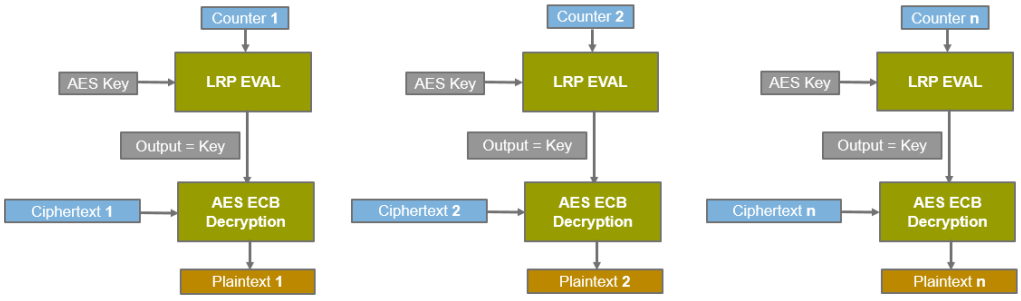


Figure 16. LRP Decryption in LRP Secure Messaging

7.2.3 LRP Precomputation Steps

The precomputation steps described in this section are needed to be executed once per key.

They are used to expand the key material by taking the key that is used for authentication, and extending it to 16 secret plaintexts and one or two updated keys.

For DESFire Light the input / output parameters are the following:

- The input key is always a 16 byte long AES key.
- The output keys (updated keys) are always 16 byte long AES keys.
- The generated plaintexts, are always 16 times 16 byte long secret plaintexts.

Two methods are used during the precomputation. One method *generatePlaintexts* for generating the secret plaintexts, as described in Section 7.2.3.1, and one method *generateUpdatedKeys* for generating the updated keys, as described in Section 7.2.3.2.

7.2.3.1 LRP Precomputation: Secret Plaintexts

The plaintext generation is needed to expand the key material and to receive 16 times 16 bytes of secret plaintext. For generating this set of secret plaintexts, the method *generateSecretPlaintexts* is used.

This function takes a key and the parameter m as input. For MIFARE DESFire Light, m is always set to the fixed value of 4 (m = 4) and the key is the key which shall be extended.

The secret plaintext generation operates in rounds. The number of rounds is specified by 2^m (being 2^4 = 16 for DESFire Light), plus adding one additional round as preparation round. Making it in total 17 round operations for DESFire Light specific calculations.

Details of the algorithm

Round 1 is a preparation step, acting as a length-doubling PRG. Round 1 takes just the 16 byte long AES key as an input and produces two keys as an output.

In Round 1, two encryption operations are executed. The round consists of the following computations:

- One encryption operation takes the input key as key, encrypts the static value 0x55 by using the standard 16 byte long zero IV and applying AES-ECB encryption.
  The output of this encryption is the new input key for Round 2.

- The other encryption operation takes the input key as key, encrypts the static value 0xAA by using the standard 16 byte long zero IV and applying AES-ECB encryption. The output of this encryption is the new input key for the function *generateUpdatedKeys*, for calculating the updated keys, as explained in Section 7.2.3.2.

Round 2 - Round 17 are computation rounds, with each round producing one 16 byte long secret plaintext. In each round, two encryption operations are executed. Each round consists of the following computations:

- One encryption operation takes the input key as key, encrypts the static value 0x55 by using the standard 16 byte long zero IV and applying AES-ECB encryption. The output of this encryption is the new input key for the next round.
- The other encryption operation takes the input key as key, encrypts the static value 0xAA by using the standard 16 byte long zero IV and applying AES-ECB encryption. The output of this encryption is the secret plaintext of this round.

The static value 0x55 is in fact a 16 byte long value, meaning 0x55555555555555555555555555555555.

The static value 0xAA is in fact a 16 byte long value, meaning 0xAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA.

The functionality of this precomputation step is depicted in Figure 17.

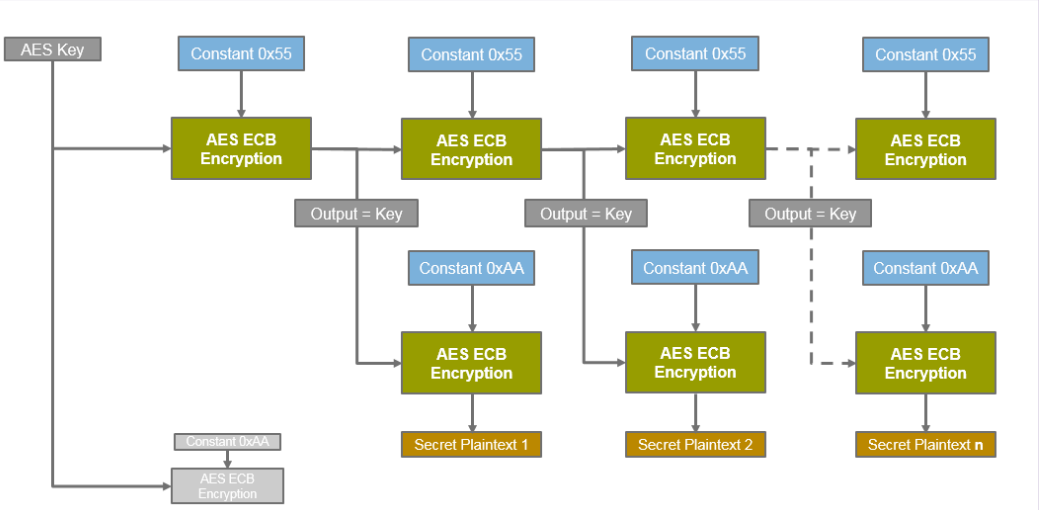


Figure 17. Generation of secret plaintexts

7.2.3.2 LRP Precomputation: Updated Keys

The generation of updated keys is needed to expand the key material and to receive a desired number of updated keys, e.g. two updated keys for MIFARE DESFire Light (one being the SessionMACKey and one being the SessionEncryptionKey). For generating the updated keys, the method *generateUpdatedKeys* is used.

This function takes a key and the parameter q as input. For MIFARE DESFire Light, q needs to be chosen with the value of 2 (q = 2) for retrieving two session keys.

As the plaintext generation, also the updated key generation operates in rounds. The number of rounds is specified by the parameter q, plus adding one additional round as preparation round.

Details of the algorithm

Round 1 is a preparation step, acting as a length-doubling PRG. Round 1 takes just the 16 byte long AES key as an input and produces two keys as an output.

In Round 1, two encryption operations are executed. The round consists of the following computations:

- One encryption operation takes the input key as key, encrypts the static value 0x55 by using the standard 16 byte long zero IV and applying AES-ECB encryption.
  The output of this encryption is the new input key for the function *generateSecretPlaintexts*, for calculating the set of secret plaintexts, as explained in Section 7.2.3.1.
- The other encryption operation takes the input key as key, encrypts the static value 0xAA by using the standard 16 byte long zero IV and applying AES-ECB encryption.
  The output of this encryption is the new input key for Round 2.

Round 2 - Round n are computation rounds, with each round producing one 16 byte long updated key. In each round, two encryption operations are executed. Each round consists of the following computations:

- One encryption operation takes the input key as key, encrypts the static value 0x55 by using the standard 16 byte long zero IV and applying AES-ECB encryption.
  The output of this encryption is the new input key for the next round.
- The other encryption operation takes the input key as key, encrypts the static value 0xAA by using the standard 16 byte long zero IV and applying AES-ECB encryption.
  The output of this encryption is the updated key of this round.

The static value 0x55 is in fact a 16 byte long value, meaning 0x55555555555555555555555555555555.

The static value 0xAA is in fact a 16 byte long value, meaning 0xAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA.

The functionality of this precomputation step is depicted in Figure 18.

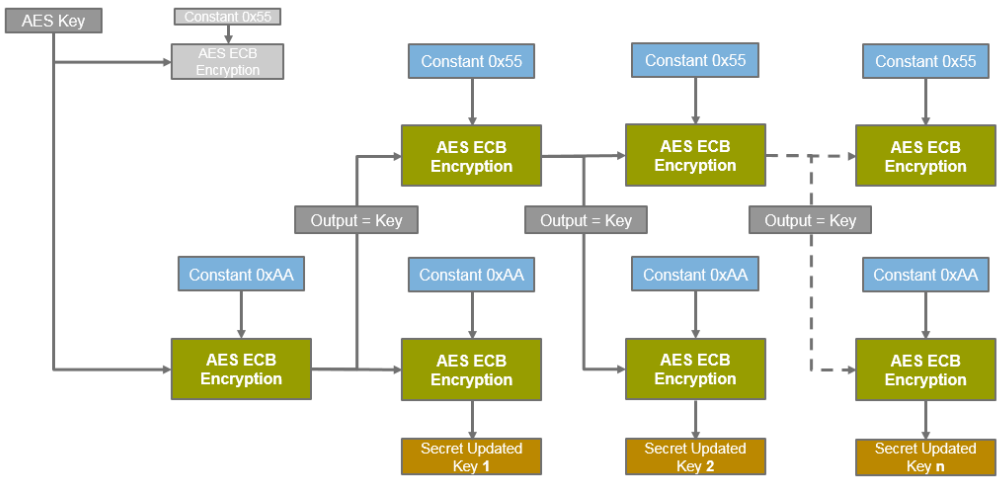


Figure 18. Generation of updated keys

7.2.4 LRP EVAL Algorithm

LRP-EVAL is the core algorithm of LRP. It uses the precomputed values (secret plaintexts and updated keys) which are explained in Section 7.2.3.

LRP-EVAL will then use one of the updated keys, depending on if a MACing or an encryption operation is requested and executed.

7.2.5 Authentication

For LRP Secure Messaging, two authentication commands Cmd.AuthenticateLRPFirst and Cmd.AuthenticateLRPNonFirst are available. The following table, Table 17, shows the differences between the two authentication commands.

Table 17. Differences between the commands AuthenticateLRPFirst and AuthenticateLRPNonFirst in LRP Secure Messaging

	Cmd.AuthenticateLRPFirst	Cmd.AuthenticateLRPNonFirst
Purpose	To start a new transaction	To start a new session within the ongoing transaction
Cmd Code	0x71	0x77
Capability Bytes	PCD and PICC capability bytes are exchanged (PDcap2, PCDcap2)	No capability bytes are exchanged
Transaction Identifier	A new transaction identifier is generated which remains valid for the full transaction	No new transaction identifier is generated (old one remains and is reused)
Command Counter	CmdCtr is reset to 0x0000	CmdCounter stays active and continues counting from the current value
Encryption Counter	EncCtr is reset to 0x00000000	EncCtr is reset to 0x00000000
Session Keys	New session keys are generated	

Cmd.AuthenticateLRPFirst can be used only if LRP secure messaging was already enabled by using the Cmd.SetConfiguration and actively setting the IC to LRP Mode. In this case, Cmd.AuthenticateLRPFirst can be used in any state (also when the PICC is in authenticated state) but Cmd.AuthenticateLRPNonFirst 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.AuthenticateLRPNonFirst. This prevents the possibility of an interleaving attack (PICC works with two PCDs at the same time) as authentication with Cmd.AuthenticateLRPNonFirst does not generate a new transaction identifier.

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

7.2.5.1 Example: Bringing the IC into LRP Secure Messaging Mode using SetConfiguration

In this example, the IC is brought into LRP mode by using the Cmd.SetConfiguration with Option 0x05.

This is an irreversible action and permanently disables AES secure messaging, meaning LRP secure messaging is required to be used for all future sessions.

Table 18. Bringing the IC to LRP Mode by using Cmd.SetConfiguration

Step	Command		Data Message
1	SetConfiguration Option	=	05
2	Session Encryption Key (SesAuthEncKey)	=	66A8CB93269DC9BC2885B7A91B9C697B
3	Session MAC Key (SesAuthMACKey)	=	7DE5F7E244A46D22E536804D07E8D70E
4	**Encrypting the Command Data**		
5	IV_Input (IV_Label || TI || Cmd Counter || Padding)	=	A55AED56F6E600000000000000000000
6	IV_Label	=	A55A
7	TI	=	ED56F6E6
8	Cmd Counter	=	0000
9	E($K_{SesAuthEnc}$, Basis for the IV))	=	DA0F644A4986275957CF1EC3AF4CCE53
10	IV	=	DA0F644A4986275957CF1EC3AF4CCE53
11	PDCap2.1	=	02
12	Data for Cmd.SetConfiguration	=	00000000020000000000
13	Padded Data	=	00000000020000000000800000000000
14	Encrypted Data = E($K_{SesAuthEnc}$, Padded Data)	=	41B2BA963075730426D0858D2AA6C498
15	**Generating the MAC for the Command APDU**		
16	IV for MACing	=	00000000000000000000000000000000

Step	Command		Data Message
17	MAC_Input (Ins || Cmd Counter || TI || Cmd Header || Encrypted Data)	=	5C0000ED56F6E60541B2BA963075730426D0858D2AA6C498
18	MAC = CMAC(KSesAuthMAC, MAC_Input)	=	2F579E77FAB49F83
19	**Constructing the full Command APDU**		
20	CLA	=	90
21	Ins	=	5C
22	P1	=	00
23	P2	=	00
24	Lc (Length of the data)	=	19
25	Data (Cmd Header || Encrypted Data || MAC)	=	0541B2BA963075730426D0858D2AA6C4982F579E77FAB49F83
26	Le (Length expected)	=	00
27	Cmd.SetConfiguration C-APDU (Cmd || Ins || P1 || P2 || Lc || Data || Le)	>	905C000019050041B2BA963075730426D0858D2AA6C4982F579E77FAB49F8300
28	Cmd Counter	=	0100
29	Cmd.SetConfiguration R-APDU	<	9100 (00 = SUCCESS)

7.2.5.2 Example: Authentication using AuthenticateLRPFirst

In this example an authentication using the Cmd.AuthenticateLRPFirst is executed, establishing successfully the LRP secure messaging. As mandatory action before being able to use Cmd.AuthenticateLRPFirst and so establishing the LRP secure messaging is setting the IC into LRP mode with the SetConfiguration command, as shown in Section 7.2.5.1.

The key number which is used for the authentication is key 0x03, an application key, with the key default value.

Key Number = 0x03

Key Value = 0x00000000000000000000000000000000

Table 19. Authentication using Cmd.AuthenticateLRPFirst

Step	Command		Data Message
1	KeyNo	=	03
2	KeyValue	=	00000000000000000000000000000000
	AuthenticateLRPFirst Part 1		
3	CLA	=	90
4	Ins	=	71
5	P1	=	00

Step	Command		Data Message
6	P2	=	00
7	Lc (Length of the data)	=	08
8	Data	=	0006020000000000
9	Le (Length expected)	=	00
10	Cmd.AuthenticateLRPFirst C-APDU (Part 1)	>	907100000800060200000000000
11	R-APDU (Part 1) = AuthMode || RndB || SW1 || SW2	<	0156109A31977C855319CD4618C9D2AED291AF
12	AuthMode	=	01
13	RndB	=	56109A31977C855319CD4618C9D2AED2
14	RndA generated by PCD	=	74D7DF6A2CEC0B72B412DE0D2B1117E6
15	RndA || RndB	=	74D7DF6A2CEC0B72B412DE0D2B1117E656109A31977C855319CD4618C9D2AED2
16	Session Vector (used for session key calculation) = fixed counter || fixed length || dynamic context || fixed label	=	0001008074D7897AB6DD9C0E855319CD4618C9D2AED2B412DE0D2B1117E69669
	AuthenticateLRPFirst Part 2		
17	Ins	=	AF
18	Data = RndA || PCDResponse	=	74D7DF6A2CEC0B72B412DE0D2B1117E689B59DCEDC31A3D3F38EF8D4810B3B4
19	PCDResponse = MAC_LRP (KSesAuthMACKey; RNDA || RNDB)	=	89B59DCEDC31A3D3F38EF8D4810B3B4
19	Cmd.AuthenticateLRPFirst C-APDU (Part 2)	>	90AF00002074D7DF6A2CEC0B72B412DE0D2B1117E6189B59DCEDC31A3D3F38EF8D4810B3B400
20	R-APDU (Part 2) = PICCData || PICCResponse || SW1 || SW2	<	F4FC209D9D60623588B299FA5D6B2D710125F8547D9FB8D572C90D2C2A14E2359100
21	PICCData = Enc_LRP (KSesAuthEncKey; TI || PDCap2 || PCDCap2)	=	F4FC209D9D60623588B299FA5D6B2D71
22	PICCData decrypted = Dec_LRP (KSesAuthEncKey; PICCData) with IV = EncCounter	=	58EE94240200000000000020000000000
23	PICCResponse = MAC_LRP (KSesAuthMacKey; RndB || RndA || PICCData)	=	0125F8547D9FB8D572C90D2C2A14E235
24	Calculated PICCResponse == Received PICCResponse	=	Yes, correct.
25	TI	=	58EE9424
26	PDcap2	=	020000000000
27	PCDcap2	=	020000000000

After the authentication, the two session keys SesAuthENCKey and SesAuthMACKey can be generated. The detailed steps for generating the session keys are explained in Section 7.2.6. The session keys always consist of one key and of the related 16 secret plaintexts. In this example, the session keys have the following values:

SesAuthENCKey = {SesAuthENCUpdateKey, 16 plaintexts} with
SesAuthENCUpdateKey = E9043D65AB21C0C422781099AB25EFDD

SesAuthMACKey = {SesAuthMACUpdateKey, 16 plaintexts} with
SesAuthMACUpdateKey = F56CADE598CC2A3FE47E438CFEB885DB

7.2.6 Session Key Generation

Already during the authentication with Cmd.AuthenticateLRPFirst or Cmd.AuthenticateLRPNonFirst, two session keys are generated both by the PICC and the PCD according to NIST SP 800-108, [8], in counter mode. The used key derivation function is the LRP-CMAC algorithm.

Two session keys are generated:

- SesAuthMACKey for MACing of APDUs, shown in Figure 19
- SesAuthENCKey for encryption and decryption of APDUs, shown in Figure 20

For LRP secure messaging, the actual session keys are not consisting of only one key, but of more key material. For constructing the session key material, it is needed to invoke several LRP-related functions which are as follows:

1. Generation of a set of secret plaintexts called AuthSPT
2. Generation of the updated key called AuthUpdateKey
3. Calculation of the authentication master key for the session, called SesAuthMasterkey, which acts as basis for deriving the SesAuthMACKey and the SesAuthENCKey. The calculation happens by using the CMAC-LRP algorithm, calculating an LRP-specific CMAC over the session vector, which is computed by using the exchanged random numbers RndA and RndB. The key used for the CMAC-LRP calculation is $K_{AuthUpdateKey}$. The initialization vector (IV) for the CMAC-LRP computation is the 16 bytes long zero IV (0x00..00).
4. Generation of a set of secret plaintexts called SesAuthSPT, which is a set of 16 times 16 byte plaintext values, associated to the two resulting session keys
5. Calculation of the two session keys SesAuthMACUpdateKey and SesAuthENCUpdateKey

The above mentioned steps can be also described in a pseudo-code style in the following way. K_x always being the key that has been used for the authentication:

1. AuthSPT = generatePlaintexts(4, K_x)
2. AuthUpdateKey = generateUpdatedKeys(1, K_x)
3. SesAuthMasterKey = CMAC-LRP($K_{AuthUpdateKey}$, SessionVector)
4. SesAuthSPT = generatePlaintexts(4, SesAuthMasterKey)
5. {SesAuthMACUpdateKey, SesAuthENCUpdateKey} = generateUpdatedKeys(2, SesAuthMasterKey)

Therefore, the SesAuthMACKey consists of the key SesAuthMACUpdateKey and also of the set of secret plaintexts SesAuthSPT.

The SesAuthENCKey consists of the key SesAuthENCUpdateKey and also of the set of secret plaintexts SesAuthSPT.

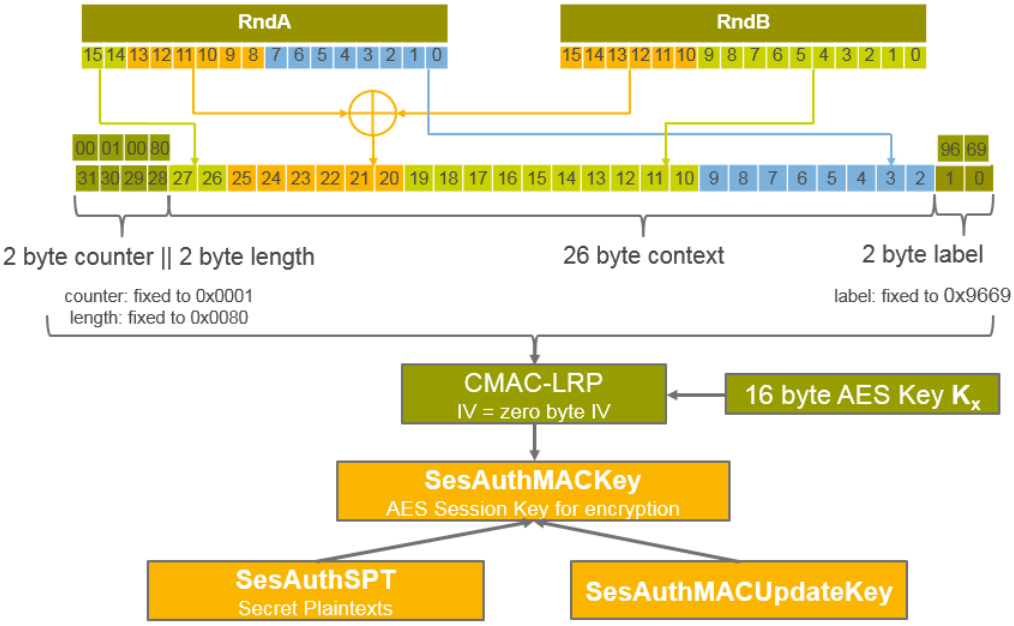


Figure 19. Session key generation for SesAuthMACKey in the LRP Secure Messaging

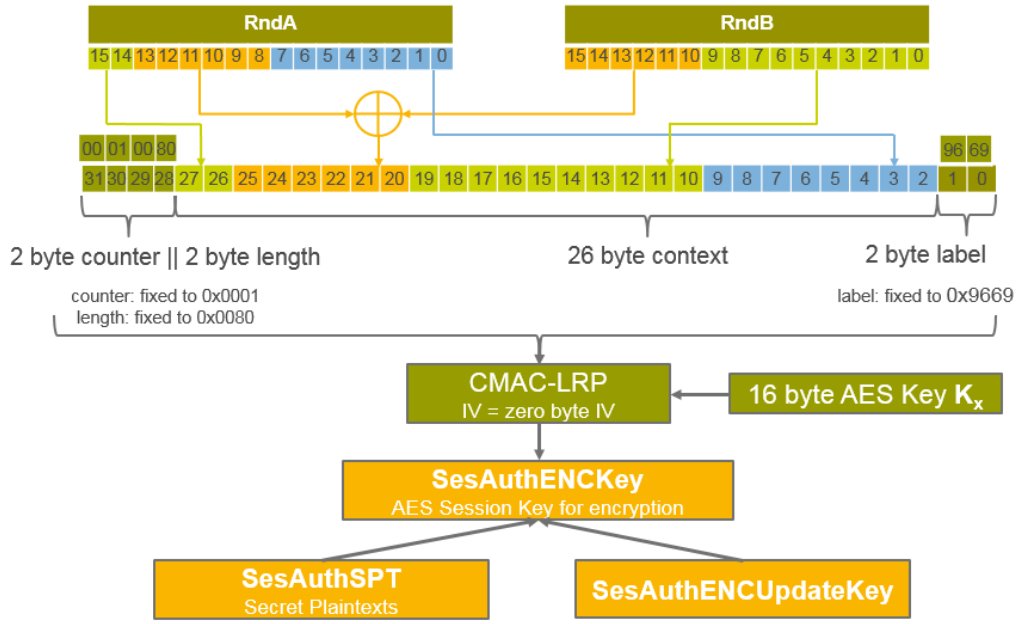


Figure 20. Session key generation for SesAuthENCKey in the LRP Secure Messaging

7.2.6.1 Example: Session Key Generation in LRP Secure Messaging

In the below shown example, the MIFARE DESFire Light specific session keys are generated.

The key that is used for the authentication is the key 0x00 with the default key value.

Table 20. Session Key Generation in LRP Secure Messaging

Step	Command		Data Message
	Key (K_x used for Authentication)	=	00000000000000000000000000000000
	Computation of the Session Vector		
1	RndA	=	74D7DF6A2CEC0B72B412DE0D2B1117E6
2	RndB	=	56109A31977C855319CD4618C9D2AED2
3	2 bytes fixed counter	=	0001
4	2 bytes fixed length tag	=	0080
5	2 bytes fixed label	=	9669
6	RndA[15::14]	=	74D7
7	RndA[13::8]	=	DF6A2CEC0B72
8	RndB[15::10]	=	56109A31977C
9	RndA[13::8] XOR RndB[15::10]	=	897AB6DD9C0E
10	RndB[9::0]	=	855319CD4618C9D2AED2
11	RndA[7::0]	=	2B412DE0D2B1117E
12	Session Vector	=	counter || length tag || RndA[15::14] || (RndA[13::8] XOR RndB[15::10]) || RndB[9::0] || RndA[7::0] || label
13	Session Vector	=	0001008074D7897AB6DD9C0E855319CD4618C9D2AED2B412DE0D2B1117E69669
	Generation of Secret Plaintexts for this Authentication (AuthSPT)		
14	AuthSPT = generatePlaintexts(4, K_x)		
15	Round 1: Pre-Step: Length-doubling PRG - Updated key for 0x55	=	AES-Encrypt: E_{Kx}(0x55555555555555555555555555555555) = 9ADAE054F63DFAFF5EA18E45EDF6EA6F
16	Round 1: Pre-Step: Length-doubling PRG - Encryption of 0xAA	=	AES-Encrypt: E_{Kx}(0xAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA) = 8522717D3AD1FBFEAFA1CEAAFDF56565
17	AuthSPT [0]	=	B5CBF983BBE3C458189436288813EC30
18	AuthSPT [1]	=	2AA01B493A7F585E10C389B5D66BD042
19	AuthSPT [2]	=	4E407179BEABA0F7ADF066B9D471E256
20	AuthSPT [3]	=	C5AC12AED5C9D18A25D6817BBAC94580
21	AuthSPT [4]	=	8BF9FF543D5F05AC7542A5900AF6D067
22	AuthSPT [5]	=	1ABFA2FF4C62CA46FF856797D0819058
23	AuthSPT [6]	=	20C7D292E3457F22C0676CA70E7B45FE
24	AuthSPT [7]	=	10C1AE93740BD443C6E15F21207B87E1

Step	Command		Data Message
25	AuthSPT [8]	=	388A3184EC11BFFFDB41FD53AC1518F8
26	AuthSPT [9]	=	09B2CF01C4D02F05352E553299E09D59
27	AuthSPT [10]	=	AF44BAD2762EC7B6DB4447A24D70B9DC
28	AuthSPT [11]	=	691F46F5E070422BD2B1740F47E0DA30
29	AuthSPT [12]	=	557A2F2F8158566164E47398ED64AFA1
30	AuthSPT [13]	=	9984E91A199ACCC2633BF95557FEA4D0
31	AuthSPT [14]	=	487F4B1959FD31BA7B2C234F27D59301
32	AuthSPT [15]	=	4EB06DF75D50712B5D20FA3700E04720
		=	
	Generation of AuthUpdateKey	=	
33	AuthUpdateKey = generateUpdatedKeys(1, K_x)		
34	Round 1: Pre-Step: Length-doubling PRG - Encryption of 0x55	=	AES-Encrypt: E_{Kx}(0x55555555555555555555555555555555) = 9ADAE054F63DFAFF5EA18E45EDF6EA6F
35	Round 1: Pre-Step: Length-doubling PRG - Updated key for 0xAA (KUpdateAA)	=	AES-Encrypt: E_{Kx}(0xAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA) = 8522717D3AD1FBFEAFA1CEAAFDF56565
36	Round 2: AuthUpdateKey	=	AES-Encrypt: $E_{KUpdateAA}$(0xAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA) = 50A26CB5DF307E483DE532F6AFBEC27B
	Generation of KSesAuthMaster		
37	KSesAuthMaster = CMAC-LRP(AuthUpdateKey, Session Vector)	=	CMAC_LRP(50A26CB5DF307E483DE532F6AFBEC27B, 0001008074d7897AB6DD9C0E855319CD4618C9D2AED2B412DE0D2B1117E69669) = 132D7E6F35BA861F39B37221214E25A5
	Generation of Secret Plaintexts for this Session (SesAuthSPT)		
38	SesAuthSPT = generatePlaintexts(4, KSesAuthMaster)		
39	Round 1: Pre-Step: Length-doubling PRG - Updated key for 0x55	=	AES-Encrypt: $E_{KSesAuthMaster}$(0x55555555555555555555555555555555) = CF60EBE3FA0EED312A687D3E6099A469
40	Round 1: Pre-Step: Length-doubling PRG - Encryption of 0xAA	=	AES-Encrypt: $E_{KSesAuthMaster}$(0xAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA) = F5C02A9B73E4D5034229C8E4A41605D1
41	SesAuthSPT [0]	=	5897B3E60A783F01856148931E9E7BAD
42	SesAuthSPT [1]	=	CB825CA9A3C44F912A9E56FE344318F7
43	SesAuthSPT [2]	=	7DD5446DC48B586C80108A528649AE26
44	SesAuthSPT [3]	=	BAE73073CC6AA5C065F47204F0CA916C
45	SesAuthSPT [4]	=	8979E6C704C8F7FC255E28F135EF55C2
46	SesAuthSPT [5]	=	2E6F796A5FE5B8936ADBAF5C39B6CFF9

Step	Command		Data Message
47	SesAuthSPT [6]	=	42DD8B4AC27B207C012BFFA95AB1C3F8
48	SesAuthSPT [7]	=	B2D717E95650207A2D50FE38376AC67C
49	SesAuthSPT [8]	=	A510D6857E87F7737B96224188C8FE36
50	SesAuthSPT [9]	=	24F8305DEB9EF312D7DBECAAB3088FFE
51	SesAuthSPT [10]	=	35FCF5E08DF53EFFE8EF2321FEA030E3
52	SesAuthSPT [11]	=	B0CAA56F89F970BF4519C7F162BD8062
53	SesAuthSPT [12]	=	5F609267603F7E881B5CE1D5C355B77D
54	SesAuthSPT [13]	=	79A407DCF45D44945D3F8205FD000AA2
55	SesAuthSPT [14]	=	C662F684EDAE28069818F3AFBCE23863
56	SesAuthSPT [15]	=	6BA505793CB49FE3F81A2E420807E9A7
	Generation of Session Keys		
57	{KSesAuthMACUpdate, KSesAuthENCUpdate} = generateUpdatedKeys(2, KSesAuthMaster)		
58	Round 1: Pre-Step: Length-doubling PRG - Encryption of 0x55	=	AES-Encrypt: $E_{KSesAuthMaster}$(0x55555555555555555555555555555555) = CF60EBE3FA0EED312A687D3E6099A469
59	Round 1: Pre-Step: Length-doubling PRG - Updated key for 0xAA (KUpdateAA)	=	AES-Encrypt: $E_{KSesAuthMaster}$(0xAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA) = F5C02A9B73E4D5034229C8E4A41605D1
60	Round 2: KSesAuthMACUpdate	=	AES-Encrypt: $E_{KUpdateAA}$(0xAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA) = F56CADE598CC2A3FE47E438CFEB885DB
61	Round 2: KUpdate55	=	AES-Encrypt: $E_{KUpdateAA}$(0x55555555555555555555555555555555) = 51079851D7AB7FB3975A3660C6937F00
62	Round 3: KSesAuthENCUpdate	=	AES-Encrypt: $E_{KUpdate55}$(0xAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA) = E9043D65AB21C0C422781099AB25EFDD
63	Round 3: KUpdate55	=	AES-Encrypt: $E_{KUpdate55}$(0x55555555555555555555555555555555) = 06ACC037D10E4E956E7279D56A2EA07A

8 Data Management

For accessing the data that is stored on the IC inside the available files, multiple data management commands, as already quickly outlined in Table 10 are available for use.

Depending on the file that shall be accessed, different commands for data exchange can be applied.

- For the Standard Data File, several examples for ReadData, WriteData as well as the ISO/IEC 7816 equivalent commands ISOReadBinary and ISOUpdateBinary, are given in Section 8.1.
- For accessing the Cyclic Record File, examples for writing and reading records are given in Section 8.2.
- For the Value File modifications using the GetValue, Credit and Debit commands, the following examples are helpful Section 8.3.

8.1 Standard Data File Operations

8.1.1 Write Data and Read Data in PLAIN communication mode

In this example, the Standard Data File with file number 0x00, configured to PLAIN communication mode, will be used for showing how the Cmd.ReadData and the Cmd.WriteData can be used.

The file with file number 0x00 is characterized through the following access conditions in an adapted configuration, after applying the Cmd.ChangeFileSettings:

- Read = 0x1
- Write = 0x1
- ReadWrite = 0x1
- ChangeConfig = 0x0

Before reading / writing an authentication with key 0x1 is necessary. After authentication, these two session keys are valid:

$K_{SesAuthMACKey}$ = 25DE813E1998D1492CADBDE939EA8D7B

$K_{SesAuthENCKey}$ = 33EED76947EF1C62310DCFB0B2727E18

Table 21. Cmd.WriteData in AES Secure Messaging using CommMode.Plain

Step	Command		Data Message
1	File No	=	00
2	Offset	=	000000
3	Data	=	22222222222222222222222222222222222222222222222222
4	DataLenght	=	190000
5	Cmd	=	90
6	Ins	=	8D
7	P1	=	00
8	P2	=	00

Step	Command		Data Message
9	Le	=	20
10	Data (FileNo || Offset || DataLenght || Data)	=	000000001900002222222222222222222222222222222222222 2222222222222
11	Lc	=	00
12	Cmd.WriteData (C-APDU)	>	908D000020000000001900002222222222222222222222222222 2222222222222222222222200
13	R-APDU	<	9100 (00 = SUCCESS)

Table 22. Cmd.ReadData in AES Secure Messaging using CommMode.Plain

Step	Command		Data Message
1	File No	=	00
2	Offset	=	000000
3	Length	=	000000 (a zero length means reading out the full file)
4	Cmd	=	90
5	Ins	=	AD
6	P1	=	00
7	P2	=	00
8	Le	=	07
9	Data (FileNo || Offset || Length)	=	00000000000000
10	Lc	=	00
11	Cmd.ReadData (C-APDU)	>	90AD0000070000000000000000
12	R-APDU	<	222222222222222222222222222222222222222222222222220 000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000 009100
13	Response Code (RC)	=	9100 (00 = SUCCESS)
14	Response Data	=	222222222222222222222222222222222222222222222222220 000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000 00

8.1.2 Write Data and Read Data in MACed communication mode

In this example, the Standard Data File with file number 0x00, configured to MACed communication mode, is used for showing how the Cmd.ReadData and the Cmd.WriteData can be used.

The file with file number 0x00 is characterized through the following access conditions in an adapted configuration, after applying the Cmd.ChangeFileSettings:

- Read = 0x1
- Write = 0x1
- ReadWrite = 0x1
- ChangeConfig = 0x0

Before reading / writing an authentication with key 0x1 is necessary. After authentication, these two session keys are valid:

$K_{SesAuthMACKey}$ = 9366FA195EB566F5BD2BAD4020B83002

$K_{SesAuthENCKey}$ = C4C9F2A734F32967FAC80A0F37C764F0

Table 23. Cmd.WriteData in AES Secure Messaging using CommMode.MACed

Step	Command		Data Message
1	File No	=	00
2	Offset	=	000000
3	Data	=	22222222222222222222222222222222222222222222222222
4	DataLenght	=	190000
	CmdCounter	=	0000
	TI	=	E2D3AF69
	Generating the MAC for the Command APDU		
	IV	=	00000000000000000000000000000000
	CmdHeader (FileNo || Offset || DataLength)	=	00000000190000
	MAC_Input (Ins || CmdCounter || TI || CmdHeader || CmdData)	=	8D0000E2D3AF690000000019000022222222222222222222222222222222222222222222
	MAC = CMAC(KSesAuthMAC, MAC_Input)	=	68F2C28C575A1628
	Constructing the full WriteData Command APDU		
5	Cmd	=	90
6	Ins	=	8D
7	P1	=	00
8	P2	=	00

Step	Command		Data Message
9	Le	=	28
10	Data (FileNo || Offset || DataLenght || Data)	=	00000000190000222222222222222222222222222222222222222222222222268F2C28C575A1628
11	Lc	=	00
12	Cmd.WriteData (C-APDU)	>	908D000028000000001900002222222222222222222222222222222222222222268F2C28C575A162800
	CmdCounter	=	0100
13	R-APDU	<	0820F68898C2A7F19100
	Response Code	=	9100 (00 = SUCCESS)
	Response MAC	=	0820F68898C2A7F1
	Verifying the received Response MAC		
	MAC_Input (RC || CmdCounter || TI)	=	000100E2D3AF69
	MAC = CMAC(KSesAuthMAC, MAC_Input)	=	0820F68898C2A7F1

Table 24. Cmd.ReadData in AES Secure Messaging using CommMode.MACed

Step	Command		Data Message
1	File No	=	00
2	Offset	=	000000
3	DataLenght	=	300000
4	CmdCounter	=	0100
5	TI	=	E2D3AF69
	Generating the MAC for the Command APDU		
6	IV	=	00000000000000000000000000000000
7	CmdHeader (FileNo || Offset || DataLength)	=	00000000300000
8	MAC_Input (Ins || CmdCounter || TI || CmdHeader || CmdData)	=	AD0100E2D3AF6900000000300000
9	MAC = CMAC(KSesAuthMAC, MAC_Input)	=	0D9BE191D5960834
	Constructing the full ReadData Command APDU		
10	Cmd	=	90
11	Ins	=	AD
12	P1	=	00

Step	Command		Data Message
13	P2	=	00
14	Le	=	0F
15	Data (FileNo || Offset || DataLenght)	=	00000000300000
16	Lc	=	00
17	Cmd.ReadData (C-APDU)	>	90AD00000F000000003000000D9BE191D596083400
18	CmdCounter	=	0200
19	R-APDU	<	22222222222222222222222222222222222222222222222220 00000000000000000000000000000000000000000000A49A44 222D9266669100
20	Response Code	=	9100 (00 = SUCCESS)
21	Response Data	=	22222222222222222222222222222222222222222222222220 00000000000000000000000000000000000000000000000
22	Response MAC	=	A49A44222D926666
	Verifying the received Response MAC		
23	MAC_Input (RC || CmdCounter || TI || ResponseData)	=	000200E2D3AF6922222222222222222222222222222222 22222222222220000000000000000000000000000000000000 000000000
24	MAC = CMAC(KSesAuthMAC, MAC_Input)	=	A49A44222D926666

8.1.3 Write Data and Read Data in FULL communication mode

In this example, the Standard Data File with file number 0x00, configured to MACed communication mode, is used for showing how the Cmd.ReadData and the Cmd.WriteData can be used.

The file with file number 0x00 is characterized through the following access conditions in an adapted configuration, after applying the Cmd.ChangeFileSettings:

- Read = 0x1
- Write = 0x1
- ReadWrite = 0x1
- ChangeConfig = 0x0

Before reading / writing an authentication with key 0x1 is necessary. After authentication, these two session keys are valid:

$K_{SesAuthMACKey}$ = 37E7234B11BEBEFDE41A8F290090EF80

$K_{SesAuthENCKey}$ = FFBCFE1F41840A09C9A88D0A4B10DF05

Table 25. Cmd.WriteData in AES Secure Messaging using CommMode.Full

Step	Command		Data Message
1	File No	=	00

Step	Command		Data Message
2	Offset	=	000000
3	Data	=	22222222222222222222222222222222222222222222222222
4	DataLenght	=	190000
5	CmdCounter	=	0000
6	TI	=	CD73D8E5
	Encrypting the Command Data		
	IV	=	00000000000000000000000000000000
11	IV_Input (IV_Label || TI || CmdCounter || Padding)	=	A55ACD73D8E500000000000000000000
12	IV for CmdData = Enc(KSesAuthENC, IV_Input)	=	871747AF36D72164A418BBFECCECD911
13	Data Block 1 (Data to write to the file)	=	22222222222222222222222222222222
14	Encrypted Data Block 1 = E(KSesAuthENC, Data Input)	=	D7446FBC912580C0A65E738D28B609E4
15	IV for CmdData	=	D7446FBC912580C0A65E738D28B609E4
16	Data Block 1 (Data to write to the file || Padding)	=	22222222222222222280000000000000
17	Encrypted Data Block 2 = E(KSesAuthENC, Data Input)	=	3ADBB8FB2B4CA68744D1BBEBB37EBD32
18	Encrypted Data (complete)	=	D7446FBC912580C0A65E738D28B609E43ADBB8FB2B4CA68744D1BBEBB37EBD32
	Generating the MAC for the Command APDU		
19	IV	=	00000000000000000000000000000000
20	CmdHeader (FileNo || Offset || DataLength)	=	00000000190000
21	MAC_Input (Ins || CmdCounter || TI || CmdHeader || Encrypted CmdData)	=	8D0000CD73D8E500000000190000D7446FBC912580C0A65E738D28B609E43ADBB8FB2B4CA68744D1BBEBB37EBD32
22	MAC = CMAC(KSesAuthMAC, MAC_ Input)	=	700ADF7BB9F62A6C
	Constructing the full WriteData Command APDU		
23	Cmd	=	90
24	Ins	=	8D
25	P1	=	00
26	P2	=	00
27	Le	=	2F

Step	Command		Data Message
28	Data (FileNo || Offset || DataLenght || Data)	=	00000000190000222222222222222222222222222222222222222222222222F2C28C575A1628
29	Lc	=	00
30	Cmd.WriteData (C-APDU)	>	908D00002F00000000190000D7446FBC912580C0A65E738D28B609E43ADBB8FB2B4CA68744D1BBEBB37EBD32700ADF7BB9F62A6C00
31	CmdCounter	=	0100
32	R-APDU	<	B9A534A7A73EE0DD9100
33	Response Code	=	9100 (00 = SUCCESS)
34	Response MAC	=	B9A534A7A73EE0DD
	Verifying the received Response MAC		
35	MAC_Input (RC || CmdCounter || TI)	=	000100CD73D8E5
36	MAC = CMAC(KSesAuthMAC, MAC_Input)	=	B9A534A7A73EE0DD

Table 26. Cmd.ReadData in AES Secure Messaging using CommMode.Full

Step	Command		Data Message
1	File No	=	00
2	Offset	=	000000
3	DataLenght	=	300000
4	CmdCounter	=	0100
5	TI	=	CD73D8E5
	Generating the MAC for the Command APDU		
6	IV	=	00000000000000000000000000000000
7	CmdHeader (FileNo || Offset || DataLength)	=	00000000300000
8	MAC_Input (Ins || CmdCounter || TI || CmdHeader || CmdData)	=	AD0100CD73D8E500000000300000
9	MAC = CMAC(KSesAuthMAC, MAC_Input)	=	7CF94F122B3DB05F
	Constructing the full ReadData Command APDU		
10	Cmd	=	90
11	Ins	=	AD
12	P1	=	00
13	P2	=	00

Step	Command		Data Message
14	Le	=	0F
15	Data (FileNo || Offset || DataLenght)	=	00000000300000
16	Lc	=	00
17	Cmd.ReadData (C-APDU)	>	90AD00000F000000003000007CF94F122B3DB05F00
18	CmdCounter	=	0200
19	R-APDU	<	8848D0F9B9FD4495770C89925B2A85C7274D350FA9029C484D43804886662DC42D7F40A6D7A415E4A71EFF79EB8E5721AC3BF1CAAFE8EB2CAA2DC162E67A97A38ED7888F22B3A5879100
20	Response Code	=	9100 (00 = SUCCESS)
21	Encrypted Response Data	=	8848D0F9B9FD4495770C89925B2A85C7274D350FA9029C484D43804886662DC42D7F40A6D7A415E4A71EFF79EB8E5721AC3BF1CAAFE8EB2CAA2DC162E67A97A3
22	Response MAC	=	8ED7888F22B3A587
	Decrypting the received Response Data		
23	IV	=	00000000000000000000000000000000
24	IV_Input_Response = 0x5A || 0xA5 || TI || CmdCtr || 0x0000000000000000	=	5AA5CD73D8E502000000000000000000
25	IV_Response = Enc(KSesAuthENC, IV_Input_Response))	=	13659A1BC863D4109FC3FB92C93D952F
26	Decrypting the Response Data = $D_{KSesAuthENC}(E_{KSesAuthENC}$(Response Data))	=	$D_{KSesAuthENC}$(8848D0F9B9FD4495770C89925B2A85C7274D350FA9029C484D43804886662DC42D7F40A6D7A415E4A71EFF79EB8E5721AC3BF1CAAFE8EB2CAA2DC162E67A97A3)
27	Decrypted Response Data	=	22222222222222222222222222222222222222222222222220000000000000000000000000000000000000000000000800000000000000000000000000000
	Verifying the received Response MAC		
28	MAC_Input (RC || CmdCounter || TI || Encrypted Response Data)	=	000200CD73D8E58848D0F9B9FD4495770C89925B2A85C7274D350FA9029C484D43804886662DC42D7F40A6D7A415E4A71EFF79EB8E5721AC3BF1CAAFE8EB2CAA2DC162E67A97A3
29	MAC = CMAC(KSesAuthMAC, MAC_Input)	=	8ED7888F22B3A587

8.1.4 Example: Reading and Writing File Content by using ISOReadBinary

In case only the inter-industry standard compliant ISO/IEC 7816-4 commands shall be used to read and modify a Standard Data File, the two commands which need to be executed are ISOReadBinary as well as ISOUpdateBinary.

In the following, a Standard Data File is selected, read out, modified and read out again. Everything is done by using only the standardized ISO/IEC 7816-4 command set.

Table 27. Select a file using ISOSelectFile

Step	Command		Data Message
1	ISOFileID	=	EF04
2	Cmd	=	00
3	Ins	=	A4
4	P1	=	00
5	P2	=	0C
6	Lc (Length of the data)	=	02
7	Data	=	EF04
8	Le	=	00
9	Cmd.ISOSelect C-APDU (ISOSelect by FileID)	>	00A4000C02EF0400
10	R-APDU	<	9000 (SUCCESS)

Table 28. Read data from a file using ISOReadBinary

Step	Command		Data Message
1	ISOFileID	=	EF04
2	Cmd	=	00
3	Ins	=	B0
4	P1	=	00
5	P2	=	00
6	Lc (Length of the data)	=	-
7	Data	=	-
8	Le	=	00 (read full file)
9	Cmd.ISOReadBinary (C-APDU)	>	00B0000000
10	R-APDU	<	00000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000 009000

Table 29. Write data to a file using ISOUpdateinary

Step	Command		Data Message
1	ISOFileID	=	EF04
2	Cmd	=	00
3	Ins	=	D6

Step	Command		Data Message
4	P1	=	00
5	P2	=	00
6	Lc (Length of the data)	=	0A
7	Data	=	11223344556677889900
8	Le	=	00
9	Cmd.ISOReadBinary (C-APDU)	>	00D600000A11223344556677889900 00
10	R-APDU	<	9000 (SUCCESS)

Table 30. Write data to a file using ISOUpdateBinary, starting at a specified offset

Step	Command		Data Message
1	ISOFileID	=	EF04
2	Cmd	=	00
3	Ins	=	D6
4	P1	=	00
5	P2	=	20
6	Lc (Length of the data)	=	12
7	Data	=	11223344556677889911223344556677 8899
8	Le	=	00
9	Cmd.ISOUpdateBinary (C-APDU)	>	00D60020121122334455667788991122334455667788990 0
10	R-APDU	<	9000 (SUCCESS)

Table 31. Read data from a file using ISOReadBinary, after content was written, starting at a specified offset

Step	Command		Data Message
1	ISOFileID	=	EF04
2	Cmd	=	00
3	Ins	=	B0
4	P1	=	00
5	P2	=	20
6	Lc (Length of the data)	=	-
7	Data	=	-
8	Le	=	00
9	Cmd.ISOReadBinary (C-APDU)	>	00B0002000

Step	Command		Data Message
10	R-APDU	<	112233445566778899112233445566778899000000000000000 000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000009000

8.2 Record File Operations

8.2.1 Example: Modifying a record file by using WriteRecord

The modification of a record file by adding a new record to it with using the WriteRecord command is shown below.

The file which is modified is FileNo 0x01, which is configured to use fully encrypted communication mode (FULL).

After an authentication with the ReadWrite key, which is in this case key number 0x01, the record can be written successfully to the file. After committing the transaction by using CommitReaderID and CommitTransaction commands, the record is added to the file permanently.

Table 32. Executing Cmd.WriteRecord in CommMode.Full

Step	Command		Data Message
1	FileNo (where the record shall be added)	=	01
2	CommMode	=	0x03 (Fully Encrypted)
3	AccessRights of the File	=	0x1012 (Read = 0x1, Write = 0x2, ReadWrite = 0x1, Change = 0x0)
4	Data (of the record that shall be written)	=	11223344556677889900112233445566
5	Length (of data of the record that shall be written)	=	10 (16 bytes)
	Authenticating using AES secure messaging		
6	KeyNo	=	01
7	KeyValue	=	01234567890123456789012345678901
8	Cmd.AuthenticateEV2First C-APDU (Part 1)	>	9071000002010000
9	R-APDU (Part 1) (E(Kx, RndB)) || SW1 || SW2 Response Code SW2 = 0xAF = Additional Frame	<	D9BE1EF2A708B27A51098AA2A39C01DF91AF

Step	Command		Data Message
10	Cmd.AuthenticateEV2First C-APDU (Part 2)	>	90AF0000202A34F282444D708D79B333D914D180E9F35CD9A83571760D0B9110A866102A6300
11	R-APDU (Part 2) E(Kx, TI || RndA' || PDcap2 || PCDcap2) || Response Code Response Code = 0x9100 = SUCCESS	<	BB9E94FCDBE27378CC0AEBFE2FF5E6FFD17FE46B5D4817405D4B0A9E99185F129100
12	TI	=	2D0611EC
13	CmdCounter (is reset to 0000 after a successful Cmd.AuthenticateEV2First command)	=	0000
14	SesAuthENCKey	=	78240CC5596B751D90023827B0B7E73D
15	SesAuthMACKey	=	6F64B98D7241EA993137DF880BEE1B89
	Encrypting the Command Data		
16	IV	=	00000000000000000000000000000000
17	IV_Input (IV_Label || TI || CmdCounter || Padding)	=	A55A2D0611EC00000000000000000000
18	IV for CmdData = Enc(KSesAuthENC, IV_Input)	=	8FDD89BAFE8C9F18CE139ABEC7C68741
19	Data (Record that shall be written to the file)	=	11223344556677889900112233445566
20	Encrypted Data = E(KSesAuthENC, Data)	=	721590E8D0C26E9F3394F33A13155378
21	IV for CmdData (= last encrypted block)	=	721590E8D0C26E9F3394F33A13155378
22	Data (Padding)	=	80000000000000000000000000000000
23	Encrypted Data = E(KSesAuthENC, Data)	=	8226661ABC3A6910AC3C7230ADF4D105
24	Encrypted Data (both blocks)	=	721590E8D0C26E9F3394F33A131553788226661ABC3A6910AC3C7230ADF4D105
		=	
	Generating the MAC for the Command APDU		
25	IV	=	00000000000000000000000000000000
26	CmdHeader (FileNo || Offset || Length)	=	01000000100000
27	MAC_Input (Ins || CmdCounter || TI || CmdHeader || Encrypted Data)	=	8B00002D0611EC01000000100000721590E8D0C26E9F3394F33A131553788226661ABC3A6910AC3C7230ADF4D105
28	MAC = CMAC(KSesAuthMAC, MAC_Input)	=	B5D409E07A0532F3

Step	Command		Data Message
	Constructing the full WriteRecord Command APDU		
29	CLA	=	90
30	Ins	=	8B
31	P1	=	00
32	P2	=	00
33	Lc (Length of the data)	=	2F
34	Data (CmdHeader || Encrypted Data || MAC)	=	0100000010000072159 0E8D0C26E9F3394F33A131553788226661ABC3A6910AC3C7230ADF4D105B5D409E07A0532F3
35	Le (Length expected)	=	00
	Exchanging the WriteRecord Command and Response APDU		
36	Cmd.WriteRecord C-APDU (Cmd || Ins || P1 || P2 || Lc || Data || Le)	>	908B00002F0100000010000072159 0E8D0C26E9F3394F33A131553788226661ABC3A6910AC3C7230ADF4D105B5D409E07A0532F300
37	CmdCounter	=	0100
38	R-APDU	<	DB552FD9D33408EF9100
39	Response Code (RC)	=	9100 (00 = SUCCESS)
40	Received MAC	=	DB552FD9D33408EF
41	MAC_Input (RC || CmdCounter || TI)	=	0001002D0611EC
42	MAC = CMAC(KSesAuthMAC, MAC_Input)	=	DB552FD9D33408EF
	Constructing the full CommitReaderID Command APDU		
43	Reader ID	=	28BF1982BE086FBC60A22DAEB66613EE
44	IV	=	00000000000000000000000000000000
45	MAC_Input (Ins || CmdCounter || TI || Data (= Reader ID))	=	C801002D0611EC28BF1982BE086FBC60A22DAEB66613EE
46	MAC = CMAC(KSesAuthMAC, MAC_Input)	=	A706434973471F1E
47	CLA	=	90
48	Ins	=	C8
49	P1	=	00
50	P2	=	00
51	Lc (Length of the data)	=	18
52	Data (Encrypted Data || MAC)	=	28BF1982BE086FBC60A22DAEB66613EEA706434973471F1E

Step	Command		Data Message
53	Le (Length expected)	=	00
	Exchanging the CommitReaderID Command and Response APDU		
54	Cmd.CommitReaderID C-APDU (Cmd || Ins || P1 || P2 || Lc || Data || Le)	>	90C800001828BF1982BE086FBC60A22DAEB66613EEA70643 4973471F1E00
55	CmdCounter	=	0200
56	R-APDU ($E_{KSesAuthENC}$(Response Data) || Response MAC || SW1 || SW2)	<	A1963F1BB9FC916A8B15B2DC58002531B11FF6023DF464E0 9100
57	Response Code (RC)	=	9100 (00 = SUCCESS)
58	Encrypted Response Data $E_{KSesAuthENC}$(Response Data)	=	A1963F1BB9FC916A8B15B2DC58002531
59	Response MAC	=	B11FF6023DF464E0
	Decrypting the received Response Data		
60	Starting IV	=	00000000000000000000000000000000
61	IV_Input_Response = 0x5A || 0xA5 || TI || CmdCtr || 0x0000000000000000	=	5AA52D0611EC02000000000000000000
62	IV_Response	=	C6565E51DDB5D2A519954CE24E93F3B7
63	Decrypting the Response Data = $D_{KSesAuthENC}$($E_{KSesAuthENC}$(Response Data))	=	$D_{KSesAuthENC}$(A1963F1BB9FC916A8B15B2DC58002531) = BDD40ED9F434F9DDCBF5821299CD2119
64	Decrypted Response Data = (TMRI)	=	BDD40ED9F434F9DDCBF5821299CD2119
65			
	Verifying the received Response MAC		
66	MAC_Input (RC || CmdCounter || TI || Encrypted Response Data)	=	0002002D0611ECA1963F1BB9FC916A8B15B2DC58002531
67	MAC = CMAC(KSesAuthMAC, MAC_ Input)	=	B11FF6023DF464E0
	Constructing the full CommitTransaction Command APDU		
68	Option (of the CommitTransaction command)	=	01 (meaning TMC and TMV to be returned in the R-APDU)
69	IV	=	00000000000000000000000000000000
70	MAC_Input (Ins || CmdCounter || TI || CmdHeader (=Option))	=	C702002D0611EC01

Step	Command		Data Message
71	MAC = CMAC(KSesAuthMAC, MAC_Input)	=	E516751A0725A624
72	CLA	=	90
73	Ins	=	C7
74	P1	=	00
75	P2	=	00
76	Lc (Length of the data)	=	09
77	Data (Option || MAC)	=	01E516751A0725A624
78	Le (Length expected)	=	00
	Exchanging the CommitTransaction Command and Response APDU		
79	Cmd.CommitTransaction C-APDU (Cmd || Ins || P1 || P2 || Lc || Data || Le)	>	90C700000901E516751A0725A62400
80	CmdCounter	=	0300
81	R-APDU (Response Data || Response MAC || SW1 || SW2)	<	0400000094A3205E41588BA9F4697772DA9DDB829100
82	Response Code (RC)	=	9100 (00 = SUCCESS)
83	Response Data (TMC || TMV)	=	0400000094A3205E41588BA9
84	Response MAC	=	F4697772DA9DDB82
85	TMC (TMAC Counter)	=	04000000
86	TMV (TMAC Value)	=	94A3205E41588BA9
	Verifying the received Response MAC		
87	MAC_Input (RC || CmdCounter || TI || Response Data)	=	0003002D0611EC0400000094A3205E41588BA9
88	MAC = CMAC(KSesAuthMAC, MAC_Input)	=	F4697772DA9DDB82

8.2.2 Example: Reading the content of a record file by using ReadRecords

How to access a record file in reading manner, with using the ReadRecords command is shown below.

The file which is accessed is FileNo 0x01, which is configured to use fully encrypted communication mode (FULL).

After an authentication with the ReadWrite key, which is in this case key number 0x01, the contents of the file can be read out successfully.

Table 33. Executing Cmd.ReadRecords in CommMode.Full

Step	Command		Data Message
1	FileNo (that shall be read)	=	01
2	CommMode	=	0x03 (Fully Encrypted)
3	AccessRights of the File	=	0x1012 (Read = 0x1, Write = 0x2, ReadWrite = 0x1, Change = 0x0)
4	RecordNo	=	000000
5	RecordCount	=	000000
	Authenticating using AES secure messaging		
6	KeyNo	=	01
7	KeyValue	=	01234567890123456789012345678901
8	TI	=	87EE66C3
9	CmdCounter (is reset to 0000 after a successful Cmd.AuthenticateEV2First command)	=	0000
10	SesAuthENCKey	=	2128E06F6A5D592E91A31535E4AB32BA
11	SesAuthMACKey	=	B0F5553474B5364FA56C2B423BFCEFCD
	Generating the MAC for the Command APDU		
12	IV	=	00000000000000000000000000000000
13	CmdHeader (FileNo || RecordNo || RecordCount)	=	01000000000000
14	MAC_Input (Ins || CmdCounter || TI || CmdHeader)	=	AB000087EE66C301000000000000
15	MAC = CMAC(KSesAuthMAC, MAC_Input)	=	180E2B1F91AB3735
	Constructing the full ReadRecords Command APDU		
16	CLA	=	90
17	Ins	=	AB
18	P1	=	00
19	P2	=	00
20	Lc (Length of the data)	=	0F
21	Data (CmdHeader || MAC)	=	010000000000000180E2B1F91AB3735
22	Le (Length expected)	=	00
	Exchanging the ReadRecords Command and Response APDU		

Step	Command		Data Message
23	Cmd.WriteRecord C-APDU (Cmd || Ins || P1 || P2 || Lc || Data || Le)	>	90AB00000F01000000000000180E2B1F91AB373500
24	CmdCounter	=	0100
25	R-APDU	<	07D38FF172A78F6908CDA660C9C585AB67314B77AA275FECDCCB34517CE32423C9E8B3726F1E83379100
26	Response Code (RC)	=	9100 (00 = SUCCESS)
27	Received MAC	=	C9E8B3726F1E8337
28	Response Data	=	07D38FF172A78F6908CDA660C9C585AB67314B77AA275FECDCCB34517CE32423
	Decrypting the received Response Data		
29	Starting IV	=	00000000000000000000000000000000
30	IV_Input_Response = 0x5A || 0xA5 || TI || CmdCtr || 0x0000000000000000	=	5AA587EE66C301000000000000000000
31	IV_Response	=	0872E68804C88499754CBA957DC95079
32	Decrypting the Response Data = $D_{KSesAuthENC}(E_{KSesAuthENC}$(Response Data))	=	$D_{KSesAuthENC}$(07D38FF172A78F6908CDA660C9C585AB67314B77AA275FECDCCB34517CE32423) = 11223344556677889900112233445566800000000000000000000000
33	Decrypted Response Data (Record Data || Padding)	=	11223344556677889900112233445566800000000000000000000000000000000000
34	Record Data	=	112233445566778899001122334455 66
	Verifying the received Response MAC		
35	MAC_Input (RC || CmdCounter || TI || Encrypted Response Data)	=	00010087EE66C307D38FF172A78F6908CDA660C9C585AB67314B77AA275FECDCCB34517CE32423
36	MAC = CMAC(KSesAuthMAC, MAC_Input)	=	C9E8B3726F1E8337

8.3 Value File Operations

8.3.1 Example: Accessing a value file by using GetValue

This example illustrates how the value which is stored in the value file can be retrieved by using the GetValue command using Plain communication mode without prior needed authentication. This means, that the setting "Free GetValue" is enabled for the file, authorizing GetValue commands without authentication.

Table 34. Executing Cmd.GetValue in CommMode.Plain (Free GetValue is enabled for the file)

Step	Command		Data Message
1	FileNo (that shall be read)	=	03

Step	Command		Data Message
2	CommMode	=	0x00 (Plain)
3	AccessRights of the File	=	0x3012 (Read = 0x1, Write = 0x2, ReadWrite = 0x3, Change = 0x0)
	Constructing the full GetValue Command APDU		
4	CLA	=	90
5	Ins	=	6C
6	P1	=	00
7	P2	=	00
8	Lc (Length of the data)	=	01
9	Data (CmdHeader || MAC)	=	03
10	Le (Length expected)	=	00
	Exchanging the GetValue Command and Response APDU		
11	Cmd.GetValue C-APDU (Cmd || Ins || P1 || P2 || Lc || Data || Le)	>	906C0000010300
12	CmdCounter	=	0100
13	R-APDU	<	000000009100
14	Response Code (RC)	=	9100 (00 = SUCCESS)
15	Response Data (Value stored in file)	=	00000000

The next example illustrates how the value which is stored in the value file can be retrieved by using the GetValue command using Full encrypted communication mode. This means, that an authentication with the Read or the ReadWrite key needs to be successfully executed before the value can be retrieved from the file.

Table 35. Executing Cmd.GetValue in CommMode.Full

Step	Command		Data Message
1	FileNo (that shall be read)	=	03
2	CommMode	=	0x03 (Fully Encrypted)
3	AccessRights of the File	=	0x3012 (Read = 0x1, Write = 0x2, ReadWrite = 0x3, Change = 0x0)
4			
5			
	Authenticating using AES secure messaging		
6	KeyNo	=	03

Step	Command		Data Message
7	KeyValue	=	00000000000000000000000000000000
8	TI	=	E412166F
9	CmdCounter (is reset to 0000 after a successful Cmd.AuthenticateEV2First command)	=	0000
10	SesAuthENCKey	=	4C4C0E575943FF670CF85BAE2E0D201D
11	SesAuthMACKey	=	D1CC5CE9FC9F1970348D33D01FAFEF8F
	Generating the MAC for the Command APDU		
12	IV	=	00000000000000000000000000000000
13	CmdHeader (FileNo)	=	03
14	MAC_Input (Ins || CmdCounter || TI || CmdHeader)	=	6C0000E412166F03
15	MAC = CMAC(KSesAuthMAC, MAC_Input)	=	B775DA280F3E7300
	Constructing the full GetValue Command APDU		
16	CLA	=	90
17	Ins	=	6C
18	P1	=	00
19	P2	=	00
20	Lc (Length of the data)	=	09
21	Data (CmdHeader || MAC)	=	03B775DA280F3E7300
22	Le (Length expected)	=	00
	Exchanging the GetValue Command and Response APDU		
23	Cmd.GetValue C-APDU (Cmd || Ins || P1 || P2 || Lc || Data || Le)	>	906C00000903B775DA280F3E730000
24	CmdCounter	=	0100
25	R-APDU	<	BC2CE0D37364B3C355C4B9B9A98802FF24035F8D39D40CE09100
26	Response Code (RC)	=	9100 (00 = SUCCESS)
27	Received MAC	=	24035F8D39D40CE0
28	Encrypted Response Data	=	BC2CE0D37364B3C355C4B9B9A98802FF
	Decrypting the received Response Data		
29	Starting IV	=	00000000000000000000000000000000

Step	Command		Data Message
30	IV_Input_Response = 0x5A || 0xA5 || TI || CmdCtr || 0x0000000000000000	=	5AA5E412166F01000000000000000000
31	IV_Response	=	4A1FBDBB8979CE3D4926F15DAD346EA7
32	Decrypting the Response Data = $D_{KSesAuthENC}(E_{KSesAuthENC}$(Response Data))	=	$D_{KSesAuthENC}$(BC2CE0D37364B3C355C4B9B9A98802FF) = 00000000800000000000000000000000
33	Decrypted Response Data (Record Data || Padding)	=	00000000800000000000000000000000
34	Data (Value)	=	00000000
35			
	Verifying the received Response MAC		
36	MAC_Input (RC || CmdCounter || TI || Encrypted Response Data)	=	000100E412166FBC2CE0D37364B3C355C4B9B9A98802FF
37	MAC = CMAC(KSesAuthMAC, MAC_Input)	=	24035F8D39D40CE0

8.3.2 Example: Modifying a value file by using the Credit command

This example illustrates how the value inside a value file can be increased by applying the Credit command. It shows the command execution in fully encrypted communication mode after a successful authentication has taken place.

Table 36. Executing Cmd.Credit in CommMode.Full

Step	Command		Data Message
1	FileNo (that shall be read)	=	03
2	CommMode	=	0x03 (Fully Encrypted)
3	AccessRights of the File	=	0x3012 (Read = 0x1, Write = 0x2, ReadWrite = 0x3, Change = 0x0)
4	Value to Credit	=	99000000 (153 in decimal)
5	TI	=	E412166F
6	CmdCounter	=	0100
7	SesAuthENCKey	=	4C4C0E575943FF670CF85BAE2E0D201D
8	SesAuthMACKey	=	D1CC5CE9FC9F1970348D33D01FAFEF8F
	Encrypting the Command Data		
9	IV	=	00000000000000000000000000000000
10	IV_Input (IV_Label || TI || CmdCounter || Padding)	=	A55AE412166F01000000000000000000
11	IV for CmdData = Enc(KSesAuthENC, IV_Input)	=	EB331BEF586438EF594A0189C0CA6DC9

Step	Command		Data Message
12	Data (Value || Padding)	=	990000008000000000000000000000000
13	Encrypted Data = E(KSesAuthENC, Data)	=	9DCDD6C409DABB0C9D5834D04FBD6E26
	Generating the MAC for the Command APDU		
14	IV	=	00000000000000000000000000000000
15	CmdHeader (FileNo)	=	03
16	MAC_Input (Ins || CmdCounter || TI || CmdHeader || Encrypted CmdData)	=	0C0100E412166F039DCDD6C409DABB0C9D5834D04FBD6E26
17	MAC = CMAC(KSesAuthMAC, MAC_Input)	=	68AFCFFD9EFB9234
	Constructing the full Credit Command APDU		
18	CLA	=	90
19	Ins	=	0C
20	P1	=	00
21	P2	=	00
22	Lc (Length of the data)	=	19
23	Data (CmdHeader || MAC)	=	039DCDD6C409DABB0C9D5834D04FBD6E2668AFCFFD9EFB9234
24	Le (Length expected)	=	00
	Exchanging the Credit Command and Response APDU		
25	Cmd.Credit C-APDU (Cmd || Ins || P1 || P2 || Lc || Data || Le)	>	900C000019039DCDD6C409DABB0C9D5834D04FBD6E2668AFCFFD9EFB923400
26	CmdCounter	=	0200
27	R-APDU	<	1004E8EA74C2871F9100
28	Response Code (RC)	=	9100 (00 = SUCCESS)
29	Received MAC	=	1004E8EA74C2871F
	Verifying the received Response MAC		
30	MAC_Input (RC || CmdCounter || TI)	=	000200E412166F
31	MAC = CMAC(KSesAuthMAC, MAC_Input)	=	1004E8EA74C2871F

8.3.3 Example: Modifying a value file by using the Debit command

This example illustrates how the value inside a value file can be reduced by applying the Debit command. It shows the command execution in fully encrypted communication mode after a successful authentication has taken place.

Table 37. Executing Cmd.Debit in CommMode.Full

Step	Command		Data Message
1	FileNo (that shall be read)	=	03
2	CommMode	=	0x03 (Fully Encrypted)
3	AccessRights of the File	=	0x3012 (Read = 0x1, Write = 0x2, ReadWrite = 0x3, Change = 0x0)
4	Value to Debit	=	71000000 (113 in decimal)
5	TI	=	E412166F
6	CmdCounter	=	0300
7	SesAuthENCKey	=	4C4C0E575943FF670CF85BAE2E0D201D
8	SesAuthMACKey	=	D1CC5CE9FC9F1970348D33D01FAFEF8F
	Encrypting the Command Data		
9	IV	=	00000000000000000000000000000000
10	IV_Input (IV_Label || TI || CmdCounter || Padding)	=	A55AE412166F03000000000000000000
11	IV for CmdData = Enc(KSesAuthENC, IV_Input)	=	612658AE21C9CB1D9F514CD46E31B3BD
12	Data (Value || Padding)	=	71000000800000000000000000000000
13	Encrypted Data = E(KSesAuthENC, Data)	=	C483B5A4C50AFFF9635F3A24F59E0A21
	Generating the MAC for the Command APDU		
14	IV	=	00000000000000000000000000000000
15	CmdHeader (FileNo)	=	03
16	MAC_Input (Ins || CmdCounter || TI || CmdHeader || Encrypted CmdData)	=	DC0300E412166F03C483B5A4C50AFFF9635F3A24F59E0A21
17	MAC = CMAC(KSesAuthMAC, MAC_Input)	=	0CA9B6B519A49F49
	Constructing the full Debit Command APDU		
18	CLA	=	90
19	Ins	=	DC
20	P1	=	00

Step	Command		Data Message
21	P2	=	00
22	Lc (Length of the data)	=	19
23	Data (CmdHeader || MAC)	=	03C483B5A4C50AFFF9635F3A24F59E0A210CA9B6B519A49F49
24	Le (Length expected)	=	00
	Exchanging the Debit Command and Response APDU		
25	Cmd.Debit C-APDU (Cmd || Ins || P1 || P2 || Lc || Data || Le)	>	90DC00001903C483B5A4C50AFFF9635F3A24F59E0A210CA9B6B519A49F4900
26	CmdCounter	=	0400
27	R-APDU	<	473A27190FDEF3439100
28	Response Code (RC)	=	9100 (00 = SUCCESS)
29	Received MAC	=	473A27190FDEF343
30			
31	**Verifying the received Response MAC**		
32	MAC_Input (RC || CmdCounter || TI)	=	000400E412166F
33	MAC = CMAC(KSesAuthMAC, MAC_Input)	=	473A27190FDEF343

9 Key Management

MIFARE DESFire Light provides you with a number of keys that can be used to grant access to the content of the MIFARE DESFire Light application in a flexible way.

On the MIFARE DESFire Light PICC level, 4 AES keys, the so-called Originality Keys, can be used for authentication to verify the authenticity of the product. These keys however are not intended to be used for granting access to any kind of information that is stored on the chip.

On the MIFARE DESFire Light Application level, 5 AES keys, the application keys, can be used for authentication toward the application itself. The keys can be used for granting access in reading / writing mode to the different files and their content.

MIFARE DESFire Light supports only one key type, namely AES. This key type can be used for authentication based on AES secure messaging as well as authentication based on LRP secure messaging.

The key type and key strength are equivalent to the AES keys which are used in MIFARE DESFire EV2. Also the AES secure messaging of MIFARE DESFire Light is fully compatible with the MIFARE DESFire EV2 secure messaging, guaranteeing perfect interoperability.

Each application key additionally is associated with a key version. The key version is set to 0x00 in the beginning and can be updated or increased when changing the key value. There is also a command for retrieving the key version from the IC, which is GetKeyVersion. An example on how this command can be used, is given in Section 9.1.

9.1 Example: Retrieve the version of a key by using GetKeyVersion

The following example shows, how the version of a key can be retrieved from the IC.

As a first step, an authentication is mandatory with the key for which the key version shall be retrieved. After successful authentication, the command GetKeyVersion for the selected key number can be sent to the card, and the currently used key version is returned from the chip.

Table 38. ChangeKey for key 0x00 after authenticating with key 0x00

Step	Command		Data Message
1	KeyNo (of which the version shall be retrieved)	=	00
	Authenticating using AES secure messaging		
2	KeyNo (used for authentication)	=	00
3	KeyValue	=	00000000000000000000000000000000
4	Cmd.AuthenticateEV2First C-APDU (Part 1)	>	9071000002000000

Step	Command		Data Message
5	R-APDU (Part 1) (E(Kx, RndB)) || SW1 || SW2 Response Code SW2 = 0xAF = Additional Frame	<	C620BC73ACC12E5F600A035C302860BB91AF
6	Cmd.AuthenticateEV2First C-APDU (Part 2)	>	90AF0000203D39B3634F6BB2E24567AABB9506D9933CA5FD9F069AF9E2A24807A6C49DE74C00
7	R-APDU (Part 2) E(Kx, TI || RndA' || PDcap2 || PCDcap2) || Response Code Response Code = 0x9100 = SUCCESS	<	23F408FF4222E644F30D3B5A65FF122976178DE7A607F08A3DD04A40BD05C63F9100
8	TI	=	5084A1A3
9	CmdCounter (is reset to 0000 after a successful Cmd.AuthenticateEV2First command)	=	0000
10	SesAuthENCKey	=	B7321D7F9E18FAA95D31651EDFDF66A9
11	SesAuthMACKey	=	AAB799EBB2B22AC79D7F3EB0E1CFD49E
	Generating the MAC for the Command APDU		
12	IV	=	00000000000000000000000000000000
13	MAC_Input (Ins || CmdCounter || TI || CmdHeader || Encrypted Data)	=	6400005084A1A300
14	MAC = CMAC(KSesAuthMAC, MAC_Input)	=	7F0A6EABC174B6DF
	Constructing the full GetKeyVersion Command APDU		
15	CLA	=	90
16	Ins	=	64
17	P1	=	00
18	P2	=	00
19	Lc (Length of the data)	=	09
20	Data (CmdHeader || MAC)	=	007F0A6EABC174B6DF
21	Le (Length expected)	=	00
	Exchanging the Command and Response APDU		
22	Cmd.GetKeyVersion C-APDU (Cmd || Ins || P1 || P2 || Lc || Data || Le)	=	9064000009007F0A6EABC174B6DF00
23	CmdCounter	=	0100

Step	Command		Data Message
24	R-APDU	=	00DF206987E53FD8C89100
25	Response Code (RC)	=	9100 (00 = SUCCESS)
26	Response MAC	=	DF206987E53FD8C8
27	Response Data (Key Version)	=	00
	Verifying the received Response MAC		
28	MAC_Input (RC || CmdCounter || TI || ResponseData)	=	0001005084A1A300
29	MAC = CMAC(KSesAuthMAC, MAC_Input)	=	DF206987E53FD8C8

9.2 Changing Application Keys

The default value for the five application keys of the MIFARE DESFire Light IC upon delivery is the key value 0x00000000000000000000000000000000 and the key version 0x00.

All the application keys of MIFARE DESFire Light can be updated / changed using the ChangeKey command, which is assigning a new key value and / or a new key version (if wanted). There are two different ways to calculate the cryptogram for the ChangeKey command, depending upon the key to be changed and key used for the currently authenticated session.

- **Case 1**: Key number to be changed ≠ Key number for currently authenticated session.
- **Case 2**: Key number to be changed == Key number for currently authenticated session.

An example for case 1 can be seen in Section 9.2.1, and another example for case 2 is illustrated in Section 9.2.2.

9.2.1 Example: Using ChangeKey in AES Secure Messaging for case 1

This example details, how a MIFARE DESFire Light application key can be changed. The key that is going to be changed is a different key of the currently used authentication key. The authentication session has been established by using a different key.

The example is referring back to **Case 1: Key number to be changed ≠ Key number used during ongoing authenticated session**.

Table 39. ChangeKey for key 0x01 after authenticating with key 0x00

Step	Command		Data Message
1	KeyNo	=	01
2	Old KeyValue	=	00000000000000000000000000000000
3	New KeyValue	=	01234567890123456789012345678901
4	Old KeyVersion	=	00
5	New KeyVersion	=	00

Step	Command		Data Message
	Authenticating using AES secure messaging		
6	KeyNo	=	00
7	KeyValue	=	00000000000000000000000000000000
8	Cmd.AuthenticateEV2First C-APDU (Part 1)	>	9071000002000000
9	R-APDU (Part 1) (E(Kx, RndB)) || SW1 || SW2 Response Code SW2 = 0xAF = Additional Frame	<	7739C880E72AFDA9CDF7DBABDFB2C87991AF
10	Cmd.AuthenticateEV2First C-APDU (Part 2)	>	90AF0000206C8A2F1C30610E1F78DE973A356F2CA8ED0134C79D8EFC1F7C5457067DE10E9A00
11	R-APDU (Part 2) E(Kx, TI || RndA' || PDcap2 || PCDcap2) || Response Code Response Code = 0x9100 = SUCCESS	<	A08AED11A84BBBBF8251E19A9DF5E00A952FB8B12F6453447452AFC8FF0C2A609100
12	TI	=	BC354CD5
13	CmdCounter (is reset to 0000 after a successful Cmd.AuthenticateEV2First command)	=	0000
14	SesAuthENCKey	=	A664EA370E90B249FE2870772291A820
15	SesAuthMACKey	=	605526E474B2304354D05A800F56B7A9
	Encrypting the Command Data		
16	IV_Input (IV_Label || TI || CmdCounter || Padding)	=	A55ABC354CD500000000000000000000
17	IV	=	00000000000000000000000000000000
18	IV for CmdData = Enc(KSesAuthENC, IV_Input)	=	6A67D1BB5A975FC3F66B54909672A64E
19	Data (New KeyValue || New KeyVersion || CRC32 of New KeyValue || Padding)	=	0123456789012345678901234567890100A0A608688000000000000000000000
20	Encrypted Data = E(KSesAuthENC, Data Input)	=	80D40DB52D5D8CA136249A0A14154DBA1BE0D67C408AB24CF0F3D3B4FE333C6A
	Generating the MAC for the Command APDU		
21	IV	=	00000000000000000000000000000000
22	MAC_Input (Ins || CmdCounter || TI || CmdHeader || Encrypted Data)	=	C40000BC354CD50180D40DB52D5D8CA136249A0A14154DBA1BE0D67C408AB24CF0F3D3B4FE333C6A

Step	Command		Data Message
23	MAC = CMAC(KSesAuthMAC, MAC_Input)	=	D27EF8006B374ABB
	Constructing the full ChangeKey Command APDU	=	
24	CLA	=	90
25	Ins	=	C4
26	P1	=	00
27	P2	=	00
28	Lc (Length of the data)	=	29
29	Data (CmdHeader || Encrypted Data || MAC)	=	0180D40DB52D5D8CA136249A0A14154DBA1BE0D67C408AB24CF0F3D3B4FE333C6AD27EF8006B374ABB
30	Le (Length expected)	=	00
		=	
	Exchanging the Command and Response APDU		
31	Cmd.ChangeKey C-APDU (Cmd || Ins || P1 || P2 || Lc || Data || Le)	=	90C40000290180D40DB52D5D8CA136249A0A14154DBA1BE0D67C408AB24CF0F3D3B4FE333C6AD27EF8006B374ABB00
32	CmdCounter	=	0100
33	R-APDU	=	BB94CB85EBC43B9D9100
34	Response Code (RC)	=	9100 (00 = SUCCESS)
35	MAC_Input (RC || CmdCounter || TI)	=	000100BC354CD5
36	MAC = CMAC(KSesAuthMAC, MAC_Input)	=	BB94CB85EBC43B9D

9.2.2 Example: Using ChangeKey in AES Secure Messaging for case 2

This example details, how a MIFARE DESFire Light application key can be changed. The key that is going to be changed is the currently used authentication key. The authentication session has been established with the same key.

The example is referring back to **Case 2: Key number to be changed == Key number used during ongoing authenticated session**.

Table 40. ChangeKey for key 0x00 after authenticating with key 0x00

Step	Command		Data Message
1	KeyNo	=	00
2	Old KeyValue	=	00000000000000000000000000000000
3	New KeyValue	=	01234567890123456789012345678901
4	Old KeyVersion	=	00
5	New KeyVersion	=	00

Step	Command		Data Message
	Authenticating using AES secure messaging		
6	KeyNo	=	00
7	KeyValue	=	00000000000000000000000000000000
8	Cmd.AuthenticateEV2First C-APDU (Part 1)	>	9071000002000000
9	R-APDU (Part 1) (E(Kx, RndB)) || SW1 || SW2 Response Code SW2 = 0xAF = Additional Frame	<	56C89455ABEE4C169A90A6CCCE26AEC891AF
10	Cmd.AuthenticateEV2First C-APDU (Part 2)	>	90AF000020762F4B07795EF384A0C72CB094CD778070CDAC940AE297AEFDC870A39BCFE47800
11	R-APDU (Part 2) E(Kx, TI || RndA' || PDcap2 || PCDcap2) || Response Code Response Code = 0x9100 = SUCCESS	<	E26E93B2F1C02F147DFA9A922417CB6FA0DAB0460428B5F4FD8FDDEB87E59F449100
12	TI	=	94297F4D
13	CmdCounter (is reset to 0000 after a successful Cmd.AuthenticateEV2First command)	=	0000
14	SesAuthENCKey	=	E156C8522F7C8DC82B0C99BA847DE723
15	SesAuthMACKey	=	45D50C1570000D2F173DF949288E3CAD
	Encrypting the Command Data		
16	IV_Input (IV_Label || TI || CmdCounter || Padding)	=	A55A94297F4D00000000000000000000
17	IV	=	00000000000000000000000000000000
18	IV for CmdData = Enc(KSesAuthENC, IV_Input)	=	BF4A2FB89311ED58E9DCBE56FC17794C
19	Data (New KeyValue || New KeyVersion || Padding)	=	01234567890123456789012345678901008000000000000000000000000000000
20	Encrypted Data = E(KSesAuthENC, Data Input)	=	BF5400DC97A1FBD65BE870716D6F11F8161BB4CA472856DB94AB94B2EC1A13E6
	Generating the MAC for the Command APDU		
21	IV	=	00000000000000000000000000000000
22	MAC_Input (Ins || CmdCounter || TI || CmdHeader || Encrypted Data)	=	C4000094297F4D00BF5400DC97A1FBD65BE870716D6F11F8161BB4CA472856DB94AB94B2EC1A13E6
23	MAC = CMAC(KSesAuthMAC, MAC_Input)	=	27CE07CF56C11091

Step	Command		Data Message
	Constructing the full ChangeKey Command APDU	=	
24	CLA	=	90
25	Ins	=	C4
26	P1	=	00
27	P2	=	00
28	Lc (Length of the data)	=	29
29	Data (CmdHeader || Encrypted Data || MAC)	=	00BF5400DC97A1FBD65BE870716D6F11F8161BB4CA472856DB94AB94B2EC1A13E627CE07CF56C11091
30	Le (Length expected)	=	00
		=	
	Exchanging the Command and Response APDU		
31	Cmd.ChangeKey C-APDU (Cmd || Ins || P1 || P2 || Lc || Data || Le)	=	90C400002900BF5400DC97A1FBD65BE870716D6F11F8161BB4CA472856DB94AB94B2EC1A13E627CE07CF56C1109100
32	CmdCounter	=	0100
33	R-APDU	=	9100
34	Response Code (RC)	=	9100 (00 = SUCCESS)

10 Transaction Management

MIFARE DESFire Light supports the grouping of data management operations into transactions. This allows for committing or aborting all previous write accesses to files with integrated backup mechanisms within the selected MIFARE DESFire Light application via a single atomic operation.

A separate command for initiating a transaction is not provided. Instead, a transaction is started by selecting the MIFARE DESFire Light application with Cmd.ISOSelect.

On top of this, MIFARE DESFire Light supports a Transaction MAC feature which allows proofing transaction execution toward a third party. This feature can be used if, for example, a merchant needs to be reimbursed by a backend for the transaction he executed. In case of tearing during a transaction (e.g.: card has been taken off, so the response of Cmd.CommitTransaction is not received), one can reactivate the PICC and readout the TMC (so-called Transaction MAC counter). The TMC is increased for every successful committed and executed transaction. If the TMC has been increased, the transaction was successful (and the calculated Transaction MAC Value, the TMV, can be used for reimbursement).

For calculating the TMV, a special Transaction MAC key is needed. This Transaction MAC key is not part of the five application keys, but stored directly in the Transaction MAC file inside the application. The key cannot be changed with a ChangeKey command, but can only be set during file creation.

As on MIFARE DESFire Light the files are already pre-created, this means that the Transaction MAC key has been also initialized with the default key value. To exchange this default key value, the Transaction MAC file is the only file which can be deleted from the IC.

If the Transaction MAC file is deleted, the TMAC feature is disabled, but the transaction management of MIFARE DESFire Light still stays intact. When creating a new Transaction MAC file, a customized TMAC key can be specified and it will be written directly into the TMAC file. From now on, this TMAC key is used for generating the TMV (Transaction MAC Value) on every successful CommitTransaction command.

This TMV can be read out from the card, and can also be returned on every successful CommitTransaction command. This value can be re-calculated by the clearing house or infrastructure backend to have a proof of successfully executed transaction. A pre-condition to do so is to have the TMAC key also stored in the backend which shall be able to re-calculate the TMV. To have the list of exchanged commands between reader terminal and card available for being able to reconstruct the transaction sequence and inputs.

In order to delete and re-create the TMAC file inside the MIFARE DESFire Light application and so to enable the TMAC feature as you would like, look at the examples Section 10.1 and Section 10.2.

10.1 Example: Delete Transaction MAC File

This example shows, how the existing TMAC file can be deleted from the application. A mandatory authentication with the application master key needs to be executed beforehand, to have the authority for file deletion.

Table 41. Executing Cmd.DeleteTransactionMACFile

Step	Command		Data Message
1	FileNo	=	0F
	Authenticating using AES secure messaging		
2	KeyNo	=	00
3	KeyValue	=	01234567890123456789012345678901
4	Cmd.AuthenticateEV2First C-APDU (Part 1)	>	9071000002000000
5	R-APDU (Part 1) (E(Kx, RndB)) || SW1 || SW2 Response Code SW2 = 0xAF = Additional Frame	<	ADE7366FB219A6F44C39C3924699D76C91AF
6	Cmd.AuthenticateEV2First C-APDU (Part 2)	>	90AF000020E76372BCF683099FB28010CE8DC9FA326766406 9262967DEC34E9855FB519F2600
7	R-APDU (Part 2) E(Kx, TI || RndA' || PDcap2 || PCDcap2) || Response Code Response Code = 0x9100 = SUCCESS	<	0FB3FE7200EA3591894FDEC3A2AAB2F5829CF622BA88BF4A 1BD3ECE29903D7B29100
8	TI	=	B350F7C9
9	CmdCounter (is reset to 0000 after a successful Cmd.AuthenticateEV2First command)	=	0000
10	SesAuthENCKey	=	CFD5F757E422144FE831842694AF69AF
11	SesAuthMACKey	=	390F25170E00278C62B718F3025FCA59
	Generating the MAC for the Command APDU		
12	IV	=	00000000000000000000000000000000
13	MAC_Input (Ins || CmdCounter || TI || CmdHeader)	=	DF0000B350F7C90F
14	MAC = CMAC(KSesAuthMAC, MAC_ Input)	=	FD3B66C63C43E2EE
	Constructing the full DeleteTransactionMACFile Command APDU		
15	CLA	=	90
16	Ins	=	DF
17	P1	=	00
18	P2	=	00
19	Lc (Length of the data)	=	09
20	Data (CmdHeader || MAC)	=	0FFD3B66C63C43E2EE

Step	Command		Data Message
21	Le (Length expected)	=	00
	Exchanging the Command and Response APDU		
22	Cmd.DeleteTransactionMACFile C-APDU (Cmd || Ins || P1 || P2 || Lc || Data || Le)	>	90DF0000090F00FD3B66C63C43E2EE00
23	CmdCounter	=	0100
24	R-APDU	<	286E234772F69CD69100
25	Response Code (RC)	=	9100 (00 = SUCCESS)
26	MAC_Input (RC || CmdCounter || TI)	=	000100B350F7C9
27	MAC = CMAC(KSesAuthMAC, MAC_Input)	=	286E234772F69CD6

10.2 Example: Create Transaction MAC File

This example shows how a new TMAC file can be created inside the MIFARE DESFire Light application. It is executed after the deletion of the TMAC file as shown in Section 10.1. The authentication was already executed with the application master key, key number 0x00.

Table 42. Executing Cmd.CreateTransactionMACFile

Step	Command		Data Message
1	FileNo	=	0F
2	New TMAC Key	=	F7D23E0C44AFADE542BFDF2DC5C6AE02
3	FileOptions (CommMode)	=	00
4	AccessRights	=	101F (Read = 0x1, Write = 0xF, ReadWrite = 0x1, Change = 0x0)
5	TMACKeyOption (AES)	=	02
6	TMACKeyVersion	=	00
7	KeyNo (used for established authentication)	=	00
8	KeyValue (of the authentication key)	=	01234567890123456789012345678901
9	TI	=	B350F7C9
10	CmdCounter	=	0100
11	SesAuthENCKey	=	CFD5F757E422144FE831842694AF69AF
12	SesAuthMACKey	=	390F25170E00278C62B718F3025FCA59
	Encrypting the Command Data		
13	IV	=	00000000000000000000000000000000

Step	Command		Data Message
14	IV_Input (IV_Label || TI || CmdCounter || Padding)	=	A55AB350F7C901000000000000000000
15	IV for CmdData = Enc(KSesAuthENC, IV_Input)	=	303DC814A2C7366D298C50DC8F90BB36
16	Data (New TMAC Key)	=	F7D23E0C44AFADE542BFDF2DC5C6AE02
17	Encrypted Data = E(KSesAuthENC, Data)	=	E64CF1262C6B798B95C950FD7353EA87
18	IV for CmdData (= last encrypted block)	=	E64CF1262C6B798B95C950FD7353EA87
19	Data (TMACKeyVersion || Padding)	=	00800000000000000000000000000000
20	Encrypted Data = E(KSesAuthENC, Data)	=	64B1F4F4C69C4068F513715C486E18B3
21	Encrypted Data (both blocks)	=	E64CF1262C6B798B95C950FD7353EA8764B1F4F4C69C4068F513715C486E18B3
	Generating the MAC for the Command APDU		
22	IV	=	00000000000000000000000000000000
23	MAC_Input (Ins || CmdCounter || TI || CmdHeader || Encrypted Data))	=	CE0100B350F7C90F00101F02E64CF1262C6B798B95C950FD7353EA8764B1F4F4C69C4068F513715C486E18B3
24	MAC = CMAC(KSesAuthMAC, MAC_Input)	=	B3BE705D3E1FB8DC
	Constructing the full CreateTransactionMACFile Command APDU		
25	CLA	=	90
26	Ins	=	CE
27	P1	=	00
28	P2	=	00
29	Lc (Length of the data)	=	2D
30	Data (CmdHeader || MAC)	=	0F00101F02E64CF1262C6B798B95C950FD7353EA8764B1F4F4C69C4068F513715C486E18B3B3BE705D3E1FB8DC
31	Le (Length expected)	=	00
	Exchanging the Command and Response APDU		
32	Cmd.CreateTransactionMACFile C-APDU (Cmd || Ins || P1 || P2 || Lc || Data || Le)	>	90DF0000090F00FD3B66C63C43E2EE00

Step	Command		Data Message
33	CmdCounter	=	0200
34	R-APDU	<	A72E11F036B2CE909100
35	Response Code (RC)	=	9100 (00 = SUCCESS)
36	MAC_Input (RC || CmdCounter || TI)	=	000200B350F7C9
37	MAC = CMAC(KSesAuthMAC, MAC_Input)	=	A72E11F036B2CE90

11 Originality Checking

The originality check allows verification of the genuineness of MIFARE DESFire Light.

Two ways are offered to check the originality of the PICC:

- Symmetric Originality Check - The first option is based on a symmetric authentication.
- Asymmetric Originality Check - The second option works on the verification of an asymmetric signature that can be retrieved from the card.

11.1 Symmetric Originality Check

Four secret symmetric Originality Keys of key type AES are present on each individual MIFARE DESFire Light IC on PICC level.

- The keys are written on the IC at the production in the NXP factory.
- Keys are created in NXP factory HSM and never leave the secure environment.
- The keys can`t be changed after the IC leaves the NXP factory.
- Originality Check is done by executing a successful LRP Authentication with one of the Originality keys. Therefore LRP mode needs to be enabled with command Cmd.SetConfiguration beforehand.
- If the authentication with one of the Originality Keys is successful, the Originality Check is successful and the authenticity of the IC is proven.

11.2 Asymmetric Originality Check

MIFARE DESFire Light contains the NXP Originality Signature, to be able to verify with a certain probability that the IC is really based on silicon manufactured by NXP Semiconductors.

Each MIFARE DESFire Light IC contains a 56 byte long elliptic curve signature, using the secp224r1 curve. The input data for signature creation is the 7 byte UID of the IC.

Signature characteristics:

- The signature is computed according to Elliptic Curve DSA (ECDSA) based on the UID of the IC.
- The asymmetric key pair (private key and public key) is created securely in NXP`s HSM. The private key remains stored in the high secure HSM inside NXP premises. The public key can be handed out.
- The resulting signature is 56 bytes long and according to SEC standard the secp224r1 curve is taken for signature creation and validation.
- A chip unique signature is embedded into each IC during manufacturing. The signature is created using the private key and signing the UID of the IC.
- The signature can be read out from the final IC, and the public key can be used to verify the signature which was embedded into the chip.

11.2.1 Originality Signature Verification

The steps for verifying the embedded Originality Signature of a MIFARE DESFire Light IC are the following:

- Activate the IC on ISO/IEC 14443-4

- Retrieve the UID from the PICC
- Retrieve the Originality Signature (56 bytes) from the PICC by using the Read_Sig command
- Verify the retrieved signature by applying an ECDSA signature verification using the corresponding public key

Originality Check public key value for MIFARE DESFire Light:
0x040E98E117AAA36457F43173DC920A8757267F44CE4EC5ADD3C54075571AEBB
F7B942A9774A1D94AD02572427E5AE0A2DD36591B1FB34FCF3D

Byte 1 of the public key, here using the value 0x04, signalizes the IETF protocol SEC1 representation of a point on an elliptic curve, which is a sequence of the fields as seen in Table 43.

The following 28 bytes represent the x coordinate of the public key.

And the last 28 bytes represent the y coordinate of the public key.

Table 43. SEC1 point representation

Field	Description
B0	{02, 03, 04}, where 02 or 03 represent a compressed point (x only), while 04 represents a complete point (x, y)
X	x coordinate of a point
Y	y coordinate of a point, optional (only present for B0 = 0x04)

The command steps and parameter details for the signature verification flow can be seen in Table 44.

Table 44. Asymmetric Originality Signature Verification

Step	Command		Data Message
1	Cmd Code	=	3C
2	Cmd Header	=	00
3	C-APDU Read_Sig	>	903C0000010000
4	R-APDU (56 bytes ECDSA signature || response code)	<	1CA298FC3F0F04A329254AC0DF7A3EB8E756C076CD1BAAF47B8BBA6DCD78BCC64DFD3E80E679D9A663CAE9E4D4C2C77023077CC549CE4A619190
5	ECDSA signature	=	1CA298FC3F0F04A329254AC0DF7A3EB8E756C076CD1BAAF47B8BBA6DCD78BCC64DFD3E80E679D9A663CAE9E4D4C2C77023077CC549CE4A61
6	UID of the IC	=	045A115A346180
	ECDSA Verification		
7	Elliptic Curve Name	=	secp224r1
8	SEC1 Point Representation	=	04

Step	Command		Data Message
9	Public Key point coordinate **xD** (28 bytes)	=	0E98E117AAA36457F43173DC920A8757267F44CE4EC5ADD3C5407557
10	Public Key point coordinate **yD** (28 bytes)	=	1AEBBF7B942A9774A1D94AD02572427E5AE0A2DD36591B1FB34FCF3D
11	Signature part 1 **r**	=	1CA298FC3F0F04A329254AC0DF7A3EB8E756C076CD1BAAF47B8BBA6D
12	Signature part 2 **s**	=	CD78BCC64DFD3E80E679D9A663CAE9E4D4C2C77023077CC549CE4A61
13	ECDSA Verification (Implementation or Cryptographic Toolset)	=	Signature valid

12 References

[1] ISO/IEC 7816-4; Identification cards -- Integrated circuit cards -- Part 4: Organization, security and commands for interchange; Publication date: April 2013; Edition 3
https://www.iso.org/standard/54550.html

[2] ISO/IEC 14443-4 Identification cards - Contactless integrated circuit cards - Proximity cards. Part 1, 2, 3, 4.

[3] Product Datasheet - MIFARE DESFire Light contactless application IC, document number 4307xx, available in NXP DocStore

[4] Product Datasheet - MIFARE DESFire EV2 contactless multi-application IC, document number 2260xx, available in NXP DocStore

[5] Application Note - AN10992 Symmetric key diversifications, document number 1653xx, available in NXP DocStore and on the NXP website
https://www.nxp.com/docs/en/application-note/AN10922.pdf

[6] NIST Special Publication 800-38A
National Institute of Standards and Technology (NIST). Recommendation for BlockCipher Modes of Operation.
http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf

[7] NIST Special Publication 800-38B
National Institute of Standards and Technology (NIST). Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication.
http://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf

[8] NIST Special Publication 800-108
National Institute of Standards and Technology (NIST). Recommendation for key derivation using pseudorandom functions.

[9] Application Note - AN12304 Leakage Resilient Primitive (LRP) Specification, document number 4660xx, available in NXP DocStore

13 Legal information

13.1 Definitions

Draft — The document is a draft version only. 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 herein and shall have no liability for the consequences of use of such information.

13.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors. In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory. Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification. Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products. NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

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.

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

Translations — A non-English (translated) version of a document is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

13.3 Trademarks

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

MIFARE — is a trademark of NXP B.V.

DESFire — is a trademark of NXP B.V.

Tables

Figures

Contents

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

Date of release: 20 January 2020
Document identifier: AN12343
Document number: 522511