SpringCard Smart Readers & Couplers

Specification of 2nd gen. master-cards

www.springcard.com

Ref. PMA16329-CA

Document identification

Category	Specifications
Classification	Restricted
Reference	PMA16329
Version	CA
State	Draft
Key words	
Summary	
Document name	[PMA16329-CA] Specification of 2nd gen Master Cards.odt
Last modification	31/07/18

Revision history

Version	Date	Author.	Valid. by		Appr. by	Remarks
			Techn.	Qualit.		
AA	19/09/16	JDA				Created from internal documentations, covering 1st gen of master-cards (Desfire EV0)
BA	30/08/17	JIZ JDA				Introducing 2nd gen of master-cards (Desfire EV1, NFC Forum type 2 / NTAG) Content of version AA becomes chapter 2.
CA	31/07/18	JDA				2nd gen of master-cards supports new options: - RSA signature (instead of ECC) - sensitive content protected by a storage key (and not only by the card's communication key) "master-card" rewritten "master-card" for improved readability.

Content

1. Introduction

1.1. Abstract

SpringCard Smart Readers and Couplers are widely configurable by the integrator or even the end-user.

The configuration is generally performed using a computer-based software, connected to the device through its main communication interface, whatever it is (USB, serial, network, BLE…). The configuration may also be performed on-the-field, as simply as waving-and keeping- a special contactless card holding the configuration in front of the device's contactless interface. Such a contactless card is named a "master-card".

1.2. Differences between 1st and 2nd generation of master-cards

The 1st generation of master-cards was based on NXP Desfire EV0 chip and were able to convey only a set configuration data. The security was based on Desfire 3DES2K protocol for authentication and secure (ciphered) communication, and HMAC-MD5 for integrity and authenticity. Such master-cards were accepted by any SpringCard device-provided that the authentication key was correct.

The 2nd generation of master-cards supports new card technologies, and provide a better security level. The cards are not only able to convey configuration data, but also a list of commands to be executed by the device, and more.

1.2.1. New card technologies

Master-cards Gen2 support the following technologies:

- NXP Desfire EV1 (or EV2),
- Host Card Emulation on Android or using a SpringCard HCE-enabled coupler.
- NXP Mifare UltraLight or NTAG for a low-cost, single-action master-card (with some restrictions, of course...),
- NXP NTAG I2C, combining the content of a NTAG with the ability to send some data to a PC or another kind of I2C master (this system is of great use in factory or after sale service).

1.2.2. Better security level

Master-cards Gen2 use a digital signature algorithm (DSA) for integrity and authenticity.

Depending on the device, the DSA algorithm could be either based on Elliptic Curve

Cryptography (ECC) or RSA, or both.

Since a unique DSA keypair could be generated easily by every customer, only a customer's master-card could change the configuration of this very customer's devices, thus preventing efficiently any kind of "denial of service" (or bad joke) attacks.

For NXP Desfire EV1 (or EV2) and HCE implementation, AES is used for authentication and secure (ciphered) communication.

Mifare UltraLight / NTAG implementations don't provide an authentication/secure communication scheme, and are therefore not suitable to convey sensitive configuration data, unless these data is encapsulated, which is limited by the small memory capacity of the card. But it remains perfectly acceptable to use such a card to change a keyboard layout or so.

1.2.3. More flexibility

Master-cards Gen2 could be used to

- Send a new configuration to the device (same as master-cards Gen1),
- Have the device execute a command or a set of commands: join or leave a network, reboot in firmware upgrade mode, run its self-test sequence...
- Have the device send back some information: serial number, firmware version…
- Send secure cryptograms to a device's SAM.

1.2.4. More versatility

Last but not least, master-cards Gen1 were supported only by Smart Readers.

Master-cards Gen2 come hand-in-hand with a new generation of SpringCard devices that are more versatile, and could switch immediately from Smart Reader to Coupler mode, and back.

Therefore, master-cards Gen2 are also processed by devices running in Coupler mode, if they are present when the device starts up.

1.3. Audience

This manual is designed for use by application developers and system integrators. It assumes that the reader has a good knowledge of computer development and security schemes.

1.4. Support and updates

Useful related materials (product datasheets, application notes, sample software, HOWTOs and FAQs…) are available at SpringCard's web site:

www.springcard.com

Updated versions of this document and others are posted on this web site as soon as they are available.

For technical support enquiries, please refer to SpringCard support page, on the web at

www.springcard.com/support

1.5. Related documents

This document uses concepts and vocabulary defined in the documentation of the "Templates", the system that allows a SpringCard Reader to process a large family of cards.

Document ref.	Description
PMA17346	Master-cards Commands
PMA13205	Smart Readers and RFID Scanners Template System

2. Master-card Gen1

2.1. Supported cards

First generation of master-cards were NXP Desfire EV0 4K.

NXP Desfire EV1 2K, NXP Desfire EV1 4K and NXP Desfire EV1 8K are acceptable replacements.

> *The security algorithm rely on the card's UID. Random-ID configuration of the Desfire cards is therefore not supported.*

2.2. Applications and files

The master-card application uses Desfire AID $_h$**504143**.

The application has two files:

- File $_h$**01** stores the configuration to be written into the Reader. The size of this file must be 512 bytes, exactly.
- File $_h$**02** stores the digital signature that proves to the Reader that the data come from a valid source. The size of this file must be 16 bytes, exactly.

2.3. Authentication and access rules

The Reader gets authenticated onto the application using key #0. The authentication method is Desfire Legacy 3DES (2K).

The Reader uses a key diversification algorithm to compute key #0 based on a Master Key and the card's UID:

- Let ***MasterAuthKey*** be a 16-B-long Master Key for Authentication,
- Let ***CardUID*** be the 7-B-long card's UID,
- Compute ***CardAuthKey = HMAC-MD5 (MasterAuthKey, CardUID)***

The card shall be formatted with key #0 = CardAuthKey.

Both file $_h$**01** and file $_h$**02** shall be readable in ciphered mode (Desfire Comm. Mode = 3) after authentication with key #0.

2.4. Content of file $_h$01

File $_h$**01** stores the configuration to be written into the Reader. The data bytes in file $_h$**01** uses the T,L,V (Tag, Length, Value) representation:

Tag is the address of the register to be written (for instance, $_h$60 for the OPT register, $_h$20 for the LKL register of Template #2, etc),

Length is the length of the Value field, expressed in bytes. Max length is 32 ($_h$20). A master-card holding a T,L,V with L>32 will be rejected. Setting length to $_h$00 deletes the current value of a register (the register retrieves its default, out-of-factory value),

Value is the content of the register.

There are a two special values:

1. Tag = $_h$FF, Length = 0 means “erase” (all registers retrieves their default, out-of-factory value). This special value should be the first entry in the file,

2. Tag = $_h$FF, Length = 7 writes a Mifare Classic key into the Reader’s Micore chipset. The 1st byte of the Value field is the address of the key, and the next 6 bytes its actual value,

Following the set of configuration entries, file $_h$01 must be filled-up by $_h$00.

2.5. Content of file $_h$02

File $_h$02 stores the digital signature that proves to the Reader that the data come from a valid source.

The signature algorithm is a HMAC, implemented as follow:

- Let ***Content*** be the content of file $_h$01 (including the $_h$00 bytes following the actual data; ***Content*** is exactly 512-B long),
- Let ***MasterSignKey*** be the 16-B-long Master Key for Signature,
- Let ***CardUID*** be the 7-B-long card’s UID,
- Compute ***CardSignKey = HMAC-MD5 (MasterSignKey, CardUID)***
- Compute ***Sign = HMAC-MD5 (CardSignKey, Content)***

The card shall be encoded with content of file $_h$02 = ***Sign***.

2.6. Reader configuration related to master-cards

2.6.1. Authentication Key

Configuration register ₕ55 stores *MasterAuthKey* together with the key number and a few options. The format of the register is the same as register AUT for Desfire, authentication EV0, as define in [PMA13205].

Specification of register ₕ55 (size 17B):

Bits	Value	Description
Byte 0		
7-6		**Communication mode**
	b00	Plain
	b01	MACed with using the session key
	b10	*RFU*
	b11	Encrypted using the session key
5-4		**Key diversification algorithm**
	b00	No diversification
	b01	Diversification using NXP RC171 algorithm
	b10	Diversification using HMAC-MD5
	b11	RFU
3-0	b0000 to b1111	**Key number within the Desfire master-card application** Must be b0000 (MasterCard uses key # 0)
Bytes 1 to 16		
		Value of the DES or 3-DES *MasterAuthKey* (16 bytes) For a DES key, both halves of the key are equal.

(Mandatory values are shown in red. Choosing another value leads to unexpected behaviour and is not supported).

2.6.2. Signature Key

Configuration register h56 stores *MasterSignKey* together with a few options.

Specification of register h56 (size 17B):

Bits	Value	Description
Byte 0		
7-6	b00	**Signature mode** *RFU, must be h00*
5-4	 b00 b01 b10 b11	**Key diversification algorithm** No diversification Diversification using NXP RC171 algorithm Diversification using HMAC-MD5 RFU
3-0	b0000	**Key number** RFU, must be b0000
Bytes 1 to 16		
		Value of the *MasterSignKey* (16 bytes) For a DES key, both halves of the key are equal.

(Mandatory values are shown in red. Choosing another value leads to unexpected behaviour and is not supported).

2.6.3. Factory keys

When a register is not explicitly configured, the Reader uses its default, factory-defined value.

- For register h55, factory value is hE0<*Factory MasterAuthKey*>.
- For register h56, factory value is h20<*Factory MasterSignKey*>.

Both values could be disclosed under NDA only.

3. Master-card Gen2 – Common features, top-level TLV structures

3.1. Overview

All the data in the master-cards are organised and managed using a Tag, Length, Value (T,L,V) principle. Most data are explicitly read by the device from the master-card's storage, where some of them (the UID of the master-card typically) come from protocol-level information.

The reading device fetches a list of TLVs from the master-card. These TLVs are divided into two groups:

- The **Public TLVs** are publicly readable (no authentication, plain communication). The reading device uses them to know whether the master-card is suitable for the device, or not.
- The **Sensitive TLVs** are likely to be protected (AES authentication and ciphered communication) when the card technology does support this feature (Desfire and HCE). They are publicly readable for card technologies that don't (Mifare UL, NTAG). The **Sensitive TLVs** are the "useful" part of the master-card: the list of configuration data, and the list of commands. They are terminated by the digital signature.

In earlier version of the specification, the "Sensitive TLVs" were named "Secure Content TLVs". This has been changed because the reader supports some common (for instance "give me your serial number") that have no impact on the security, and it was therefore allowed to store them in plain on a Mifare UL or NTAG.

The current version of the specification uses the term "Sensitive TLVs" and makes provision to store them in a secure way even on an unsecure card such as Mifare UL or NTAG.

It is now the master-card creation software's responsibility to decide whether these "Sensitive TLVs" could safely be store in plain or not. The "Security flags TLV" has been introduce to cover the various use cases.

3.2. Format of TLVs

A TLV object has three fields:

- **T (Tag)**: this is the operation-code of a command, or the identifier of a data field. Its length is **1 byte**.
- **L (Length)**: this is the length of the following V field. **L is on 1 or 3 bytes**:
  - ➔ If length of V field is between $_h$`00` and $_h$`80`, L field is on 1 byte,
  - ➔ If length of V field is greater than $_h$`80`, L field is on 3 bytes: 1st byte is $_h$`82`, 2nd and 3rd byte hold the 16-bit value, in big endian order,
- **V (Value)**: the data field. The exact length is the value of L.

The V field of a TLV may contain several other TLV objects.

3.3. List of Tags

Tag	Length	Type	Description	See §
h`00`	0	-	Terminator – denotes the end of the list of TLVs	
h`01`	Var.	Scalar	Card UID	3.4.1
Public TLVs				
h`10`	2	Scalar	Brand ID	3.4.2
h`11`	4	Scalar	Key ID	3.4.3
h`12`	4	Scalar	Vendor ID / Product ID	3.4.4
h`13`	1	Scalar	Operating mode	3.4.5
h`14`	4 or 6	Scalar	Serial number	3.4.6
Sensitive TLVs				
h`20`	Var.	List	List of commands	3.5.1
h`40`	Var.	List	List of configuration data	3.5.2
h`50`	Var.	List	List of APDUs to send to the SAM	3.5.3
h`70`	16	Scalar	AES CMAC digital signature	7.2
h`71`	16	Scalar	ECC digital signature using curve secp128r1	7.3.1
h`72`	32	Scalar	ECC digital signature using curve secp256r1	7.3.2
h`73`	128	Scalar	1024-bit RSA digital signature	7.4.1
h`74`	256	Scalar	2048-bit RSA digital signature	7.4.2

The master-cards must contain at least one signature.

If case of a master-card containing more than one signature, the reader chooses freely the one it wants to process.

The choice between the different signature algorithms is discussed in § 7.1.

3.4. Public TLVs

3.4.1. master-card's UID

The Tag $_h$**01** is used only when computing the signature, and does not appear anywhere in the storage.

See chapter 7 "Master-card Gen2 – Digital signature" for details.

3.4.2. Brand ID

The Tag $_h$**10** is used to discriminate between the devices developed by SpringCard in white brand for various OEMs. Every brand has a unique, 16-bit ID (assigned by SpringCard). The master-cards created for one brand will be discarded by the devices belonging to another.

The absence of the **Brand ID** Tag denotes a device sold under the SpringCard brand. This is equivalent to setting the Brand ID to $_h$**0000**.

3.4.3. Key ID

The Tag $_h$**11** is used to discriminate between devices bought by different customers.

Every customer is responsible for generating is own set of keys (private and public keys, and authentication key).

The customer's **Key ID** is typically the CRC32 of the customer's public key (but may be freely chosen arbitrary).

There are two reserved values:

- Key ID = $_h$**00000000** denotes the use of SpringCard's default (transport) key set; the corresponding keys are known by all SpringCard's configuration software such as MultiConf,
- Key ID = $_h$FFFFFFFF denotes the use of SpringCard's "erase-all" key set, used by after-sale service to recover a device whose keys are unknown.

3.4.4. Vendor ID / Product ID

The Tag $_h$**12** is used to discriminate between devices based on their firmware identifier. SpringCard uses a firmware identifier compliant with the USB specification: 16-bit Vendor ID, followed by 16-bit Product ID.

Unless requested by a particular OEM for a white brand device, all devices developed by SpringCard have Vendor ID = $_h$**1C34**. A different Product ID is assigned for every firmware.

Note that the same product, running the same firmware, is likely to have a different Product ID depending on its current operating mode. This is a requirement of USB (because a different operating mode generally involves a different USB driver, and most operating systems rely on the Vendor ID + Product ID to select the driver to load).

3.4.5. Operating mode

The Tag $_h$**13** is used to discriminate between devices based on their operating mode.

Registered Operating modes are:

Mode	Description
$_h$01	Coupler, legacy SpringProx protocol, serial communication
$_h$02	Coupler, PC/SC (USB CCID protocol or alike)
$_h$03	Smart Reader, keyboard emulation (USB HID protocol or alike)
$_h$07	Smart Reader, proprietary protocol (non HID)

All other values are RFU and shall not be used until specified.

This allow a master-card to target “any device running in keyboard emulation mode” or “any device running in PC/SC mode”, where the Vendor ID + Product ID targets only a more restrictive device family.

3.4.6. Serial number

The Tag $_h$**14** is used to discriminate between devices based on their (unique) serial number.

The serial number is 6-byte long for network-attached (MAC address) and Bluetooth (BT_ADDR) devices. It is 4-byte long for USB or serial only devices.

3.5. Sensitive TLVs

3.5.1. List of commands

The Tag $_h$`20` groups all the commands to be executed by the device. All the commands are concatenated in the V field. L indicates the total length of all the concatenated commands. The device executes the commands in the order they are found in the V field.

Every command is itself stored as a TLV.

Commands are documented in [PMA17346].

It is assumed that a master-card programmer knows exactely what he/she is doing, as several commands may not be compatible with each other, or with the targeted device.

3.5.2. List of configuration data

The Tag $_h$`40` Tag groups all the registers values that need to be set (or erased). This is the same as the content of file $_h$`01` in Master-card Gen1 (see § 2.4)

The V field contains a list of TLV objects, each of them pertaining to a special register, with the following rules :

- T is the register number
- L field indicates the length of V field
- V field host the new value of the register

To erase a register while preserving the others, set L to $_h$00 (V empty).

3.5.3. List of APDUs to send to the SAM

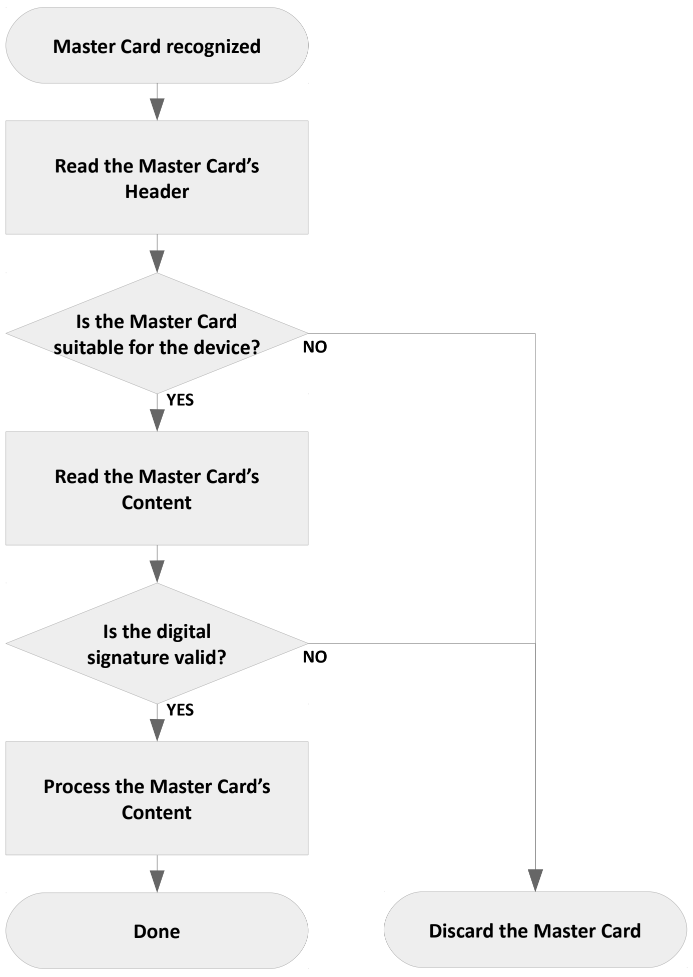
The Tag $_h$`50` Tag groups all the APDUs to be sent to the device's SAM (if some).

3.5.4. Digital signature

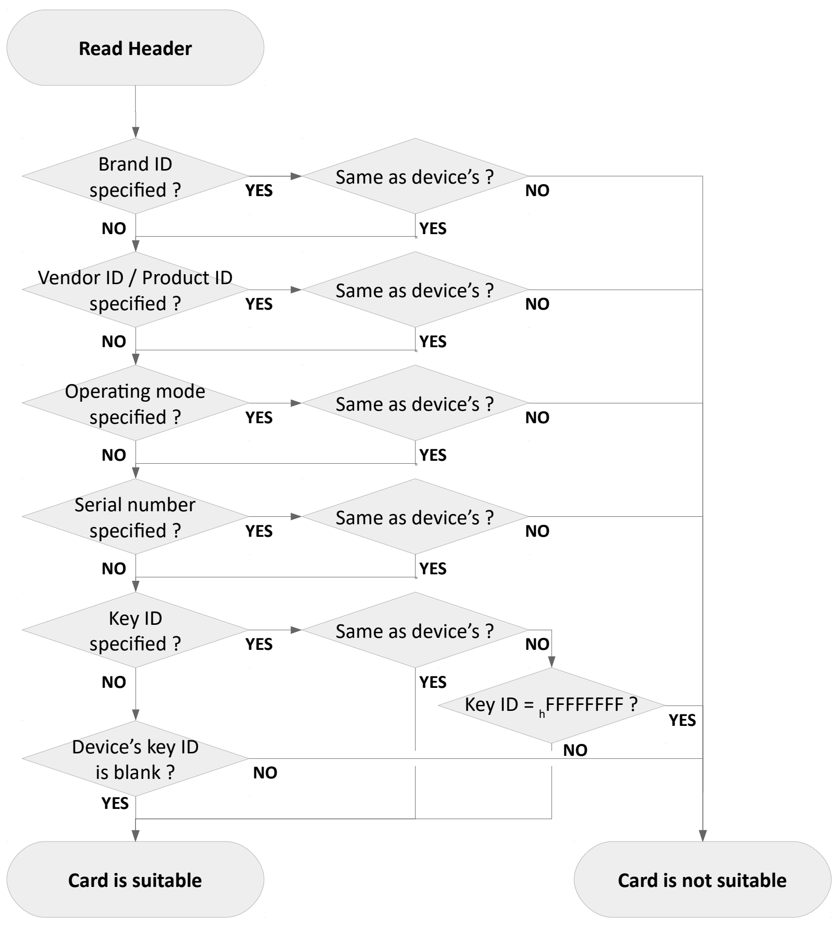
See chapter 7 "Master-card Gen2 – Digital signature" for details.

3.6. Workflow

3.6.1. General workflow


Master Card recognized
Read the Master Card's Header
Is the Master Card suitable for the device?
NO
YES
Read the Master Card's Content
Is the digital signature valid?
NO
YES
Process the Master Card's Content
Done
Discard the Master Card

3.6.2. Is the master-card suitable for the device?



3.6.3. Is the digital signature valid?

See chapter 7 “Master-card Gen2 – Digital signature” for details.

4. Master-card Gen2 – Mifare UL/NTAG implementation

4.1. Preliminary note – Security warning

The Mifare UltraLight / NTAG cards do not support any kind of cryptographic feature. There's no authentication and no ciphering. Therefore, no data confidentiality can be ensured.

As a consequence, these cards SHALL NOT BE USED to change sensitive or confidential data in the device (access keys for instance).

4.2. Memory mapping

To be used as a 2nd gen. master-card, a Mifare UltraLight / NTAG shall be formatted according to NFC Forum Type 2 Tag specification; both the Public Content and Secure Content (including the signature) are encapsulated in a single NDEF record.

4.2.1. NFC Forum type 2 encapsulation

NTAG cards and NTAG I2C chips are delivered with a NFC Forum type 2 header. Therefore, the master-card is encapsulated into a NDEF message. The page 3 (OTP) of the Card must also contain a valid NFC Forum Capability Container (CC).

Parameter	**Value**
NDEF TNF	External RTD ($_h$`04`)
NDEF Type	**`urn:com.springcard:masterv2`**

4.2.2. Content

Since the Mifare UltraLight / NTAG don't support authentication and ciphered communication, both the Public content and Sensitive content are stored one after the other in the NDEF record, and publicly readable.

4.3. Digital signature

The digital signature is computed over the protocol-level UID and the Content. Random ID is not supported for this card technology. The NFC Forum headers and fields are not included in the signature.

5. Master-card Gen2 – Desfire implementation

5.1. Memory mapping

5.1.1. Application ID

The application AID for SpringCard Master-card Gen2 is: **${}_{h}$4D4332**.

5.1.2. Public content file

The Public TLVs are stored in file ${}_{h}$01. This file is mandatory. Its parameters are:

Parameter	Value
File ID	${}_{h}$01
Type	Standard
Size	Min 64 bytes
Communication mode	Plain
Access conditions	Read: free Write, manage: after authentication with key ${}_{h}$00

If the size of the file is greater than the actual size of its content, it must be filled up with ${}_{h}$00.

5.1.3. Sensitive file

The Sensitive TLVs are stored in file ${}_{h}$02. This file is mandatory. Its parameters are:

Parameter	Value
File ID	${}_{h}$02
Type	Standard
Size	Min 64 bytes
Communication mode	Secured (cipher + CMAC)
Access conditions	Read: after authentication with key ${}_{h}$01 or key ${}_{h}$00 Write, manage: after authentication with key ${}_{h}$00

If the size of the file is greater than the actual size of its content, it must be filled up with ${}_{h}$00.

5.1.4. Responses file

The device provides its responses to the commands by adding records into file $_h$03. This file is optional. Its parameters are:

Parameter	Value
File ID	$_h$03
Type	Cyclic
Size	64 bytes per record Min 2 records
Communication mode	Plain
Access conditions	Read, write: after authentication with key $_h$01 or key $_h$00 Manage: after authentication with key $_h$00

How the Responses file is used by the device is detailed in § 5.4.

5.2. Security and access keys

5.2.1. Key $_h$00 (administration)

Parameter	Value
Key ID	$_h$00
Type	AES
Access rights	Full

This key belongs to the application that creates the master-cards. It is not used by the devices, and therefore does not have to be transmitted to the staff installing the devices.

It is recommended that this key is diversified, based on the card's UID.

5.2.2. Key $_h$01 (read)

Parameter	Value
Key ID	$_h$01
Type	AES
Access rights	Read file $_h$02, insert record in file $_h$03

This key shall be inserted into the devices when commissioning them.

This key shall not be diversified[1].

[1] If the card uses a random ID, the authentication with Key $_h$01 allows to retrieve the actual UID, which is needed to verify the signature, or to get authenticated with the Key $_h$00 if diversification applies.

5.3. Digital signature

The digital signature is computed over the actual Card UID and the Content.

If the card uses Random ID, the actual UID is read after the authentication using key $_h$01.

5.4. How the device sends its responses

When the device has a response to send back, the behaviour is as follow:

1. Send command Write Record into record 0 of the Response file
2. Send command Commit Transaction

This is performed in a loop while in case there are more than one response to send. Since the Response file has the cyclic type, the responses will be queues in the list of records.

It's the responsibilities of the application that creates the master-cards to ensure that the number of commands that raise a response remains lower than the number of records in the Responses file.

6. Master-card Gen2 – Host Card Emulation implementation

6.1. Overview

The HCE implementation is an overload of the Desfire implementation, with a slight difference: the device is able to “tell” the card “what” it is, so the HCE application may provide a different content for every device.

6.2. Workflow

1. The device selects the application and receives the master-card’s UID in return (§ 6.3),
2. The device sends its information to the master-card by writing the Device Information file,
3. The master-card populates its two Content files accordingly,
4. The device retrieves the Public content and checks that it is acceptable,
5. The device gets authenticated,
6. The device retrieves the Sensitive content and verifies the digital signature,
7. The device processes the master-card’s Content. Response could be sent back the master-card through the Response File.

6.3. ISO 7816-4 selection

6.3.1. Select Application Command

Field	Value	Details
CLA	$_h$00	
INS	$_h$A4	
P1	$_h$04	
P2	$_h$00	
L_C	$_h$0E	
DataIn	$_h$A0000006144D41535445524 32E32	

6.3.2. Select Application Response

Field	Value	Details
DataOut	`XX..XX` (16 bytes)	Card ID[2]
SW	$_h$`9000`	

6.4. Desfire emulation

The HCE implementation is an overload of the Desfire implementation (chapter 5).

The Desfire commands shall be implemented using the ISO 7816-4 encapsulation.

6.4.1. Application ID

See 5.1.1.

6.4.2. Device Information file

The Device Information TLVs are written by the device in file $_h$00. This file is mandatory. Its parameters are:

Parameter	Value
File ID	$_h$00
Type	Standard
Size	Min 64 bytes
Communication mode	Plain
Access conditions	Write: free Read, manage: handled by the HCE application

If the size of the file is greater than the actual size of its content, it must be filled up with $_h$00.

[2] The Card ID may be random or fixed, depending on the implementation. This has no impact on the rest of the transaction, provided that the digital signature is computed over this ID.

6.4.3. Public content file

The Public TLVs are stored in file $_h$01. This file is mandatory. Its parameters are:

Parameter	Value
File ID	$_h$01
Type	Standard
Size	Min 64 bytes
Communication mode	Plain
Access conditions	Read: free Write, manage: handled by the HCE application

If the size of the file is greater than the actual size of its content, it must be filled up with $_h$00.

6.4.4. Sensitive content file

The Sensitive TLVs are stored in file $_h$02. This file is mandatory. Its parameters are:

Parameter	Value
File ID	$_h$02
Type	Standard
Size	Min 64 bytes
Communication mode	Secured (cipher + CMAC)
Access conditions	Read: after authentication with key $_h$01 Write, manage: handled by the HCE application

If the size of the file is greater than the actual size of its content, it must be filled up with $_h$00.

6.4.5. Responses file

The device provides its responses to the commands by adding records into file $_h$03. This file is optional. Its parameters are:

Parameter	Value
File ID	$_h$03
Type	Cyclic
Size	64 bytes per record Min 2 records
Communication mode	Plain
Access conditions	Read, write: after authentication with key $_h$01 or key $_h$00 Manage: after authentication with key $_h$00

How the Responses file is used by the device is detailed in § 5.4.

6.5. Security and access keys

6.5.1. Key $_h$01 (read)

This key shall be the same for HCE cards as for Desfire card. See § 5.2.2.

6.6. Digital signature

The digital signature is computed over the Card ID and the Content.

6.7. How the device sends its responses

See 5.4.

7. Master-card Gen2 – Digital signature

7.1. Principles

Master-cards Gen2 supports 3 signature schemes:

- AES based CMAC,
- Elliptic curve digital signature (ECC-DSA),
- RSA digital signature (RSA-DSA).

7.1.1. How is the message constructed?

For Desfire and HCE, the message to be signed is made by concatenating the contents of both the Public file and the Sensitive files, as if they where belonging to a single file. For Mifare UL/NTAG, the message is the complete content itself.

In the case of the AES based CMAC, the CMAC is computed over the master-card's content only, the master-card's UID being fed as diversification input to the AES key.

In the case of ECC-DSA and RSA-DSA, the content is prefixed by the master-card's UID, then a SHA256 hash is extracted, and the signature is computed over this hash.

7.1.2. Which digital signature algorithm shall be chosen?

Not all devices implements the 5 algorithms, depending on the intrinsic limitations of the MCU and the availability of a secure element (NXP SAM AV2) in the device.

Also, the 5 types of signatures could not be stored in the master-cards, because it would require too much memory and take too long to be read.

7.1.2.1. Decision tree in the device

Tag $_h$74 present and the device supports RSA2048?

→ verify the RSA2048 signature, skip others

Tag $_h$72 present and the device supports ECC32?

→ verify the ECC32 signature, skip others

Tag $_h$73 present and the device supports RSA1024?

→ verify the RSA1024 signature, skip others

Tag $_h$71 present and the device supports ECC16?

→ verify the ECC16 signature, skip others

Tag $_h$70 present and the device configuration allows to accept AES CMAC?

→ verify the AES CMAC signature

otherwise discard the master-card.

7.1.2.2. Decision tree in the master-card creation software

All target devices support RSA2048?

→ Provide a RSA2048 signature in tag $_h$74, exit here

All target devices support ECC32?

→ Provide an ECC32 signature in tag $_h$72, exit here

All target devices support RSA1024?

→ Provide a RSA1024 signature in tag $_h$73, exit here

All target devices support ECC16?

→ Provide an ECC16 signature in tag $_h$71, exit here

otherwise provide an AES CMAC in tag $_h$70.

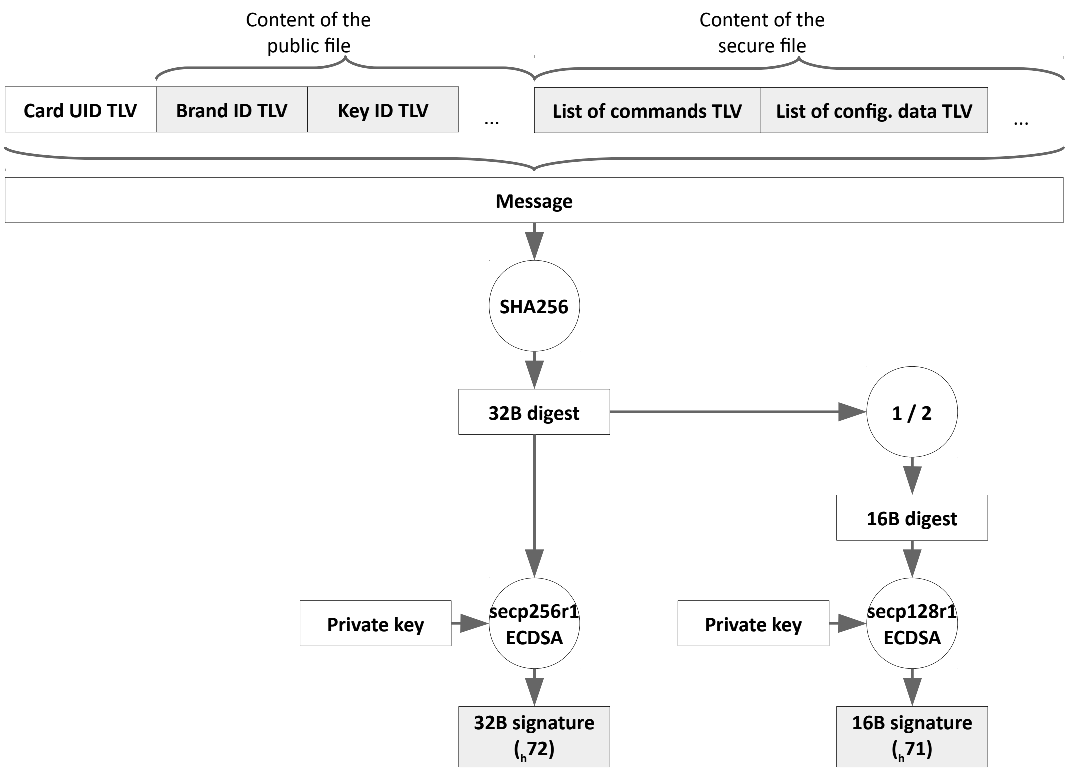
7.2. Digital signature using AES CMAC

7.3. Digital signature using ECC-DSA

Two curves are specified:

- The SECP 128 R1 curve uses 16-byte keys, and produces a 16-byte signature.
- The SECP 256 R1 curve uses 32-byte keys, and produces a 32-byte signature.

The implementation in the devices depends on the MCU. 16-bit MCUs (such as the K663's Renesas RL78) supports only 16-byte signatures. 32-bit MCUs (such as the X518's Renesas RX65x) supports only 32-byte signatures.



7.3.1. 16-byte Signature (Tag = $_{h}$71)

7.3.1.1. Curve

The algorithm is ECC-DSA. The curve is secp128r1 (“SECP 128 R1”):

Parameter	Value
p	$_{h}$FFFFFFFDFFFFFFFFFFFFFFFFFFFFFFFF
a	$_{h}$FFFFFFFDFFFFFFFFFFFFFFFFFFFFFFFC
b	$_{h}$E87579C11079F43DD824993C2CEE5ED3
x	$_{h}$161FF7528B899B2D0C28607CA52C5B86
y	$_{h}$CF5AC8395BAFEB13C02DA292DDED7A83
order	$_{h}$FFFFFFFE0000000075A30D1B9038A115

On this curve,

- The private key is a 16-byte value,
- The public key is a pair of 16-byte values,
- The signature is a 16-byte value.

7.3.1.2. Signature generation

Let **message** be the message to be signed.

1. Compute **digest32 = SHA256 (message)**
2. Extract **digest16 = even-order bytes of digest32**
3. Compute **signature = ECC-DSA-Sign (secp128r1, private key, digest16)**

7.3.1.3. Signature verification

1. Compute **digest32 = SHA256 (message)**
2. Extract **digest16 = even-order bytes of digest32**
3. Run **ECC-DSA-Verify (secp128r1, public key, digest16)**

7.3.2. 32-byte Signature (Tag = $_h$72)

7.3.2.1. Curve

The algorithm is ECC-DSA. The curve is secp256r1 (“SECP 256 R1”):

Parameter	Value
p	$_h$FFFFFFFF00000001000000000000000000 00000000FFFFFFFFFFFFFFFFFFFFFFFF
a	$_h$FFFFFFFF00000001000000000000000000 00000000FFFFFFFFFFFFFFFFFFFFFFFC
b	$_h$5AC635D8AA3A93E7B3EBBD55769886BC 651D06B0CC53B0F63BCE3C3E27D2604B
x	$_h$6B17D1F2E12C4247F8BCE6E563A440F2 77037D812DEB33A0F4A13945D898C296
y	$_h$4FE342E2FE1A7F9B8EE7EB4A7C0F9E16 2BCE33576B315ECECBB6406837BF51F5
order	$_h$FFFFFFFF00000000FFFFFFFFFFFFFFFF BCE6FAADA7179E84F3B9CAC2FC632551

On this curve,

- The private key is a 32-byte value,
- The public key is a pair of 32-byte values,
- The signature is a 32-byte value.

7.3.2.2. Signature generation

Let **message** be the message to be signed.

1. Compute **digest = SHA256 (message)**
2. Compute **signature = ECC-DSA-Sign (secp256r1, private key, digest)**

7.3.2.3. Signature verification

1. Compute **digest = SHA256 (message)**
2. Run **ECC-DSA-Verify (secp256r1, public key, digest)**

7.4. Digital signature using RSA-DSA

7.4.1. RSA 1024 (Tag = $_h$73)

7.4.2. RSA 2048 (Tag = $_h$74)

8.

8.1.

EDITOR'S INFORMATION

SPRINGCARD SAS company with a capital of 227 000 €

RCS EVRY B 429 665 482

Parc Gutenberg, 2 voie La Cardon

91120 Palaiseau – FRANCE

CONTACT INFORMATION

For more information and to locate our sales office or distributor in your country or area, please visit

www.springcard.com