A recent open source embedded implementation of the DESFire specification designed for on-the-fly logging with NFC based systems

Maxie Dion Schmidt

Georgia Institute of Technology, Atlanta, GA 30318, USA
mschmidt34@gatech.edu
WWW home page: http://people.math.gatech.edu/~mschmidt34
Public software profile: https://github.com/maxieds

Abstract. The Chameleon Mini is a portable device that interfaces with the NFC protocol over RFID. This device is designed to facilitate on-the-fly logging of data exchanges between contactless cards and tag readers operating over NFC in the 13.56MHz band. It is an indispensible tool for researchers, reverse engineers and system penetration testers that perform security analysis over the protocol. The Chameleon Mini also supports emulation of many contactless card types over NFC that are enabled by contributions to its open source embedded firmware. In 2020, we set out to offer a fully functional open source implementation to provide a frequently requested interface to the complex and proprietary DESFire tag command set and internal architecture specification from within the open source firmware for the Chameleon Mini RevG. In this application note, we describe the technologies utilized, applications of this work, and describe the challenges of the low-level implementation of the embedded software.

Keywords: Near field communication, NFC technology, Chameleon Mini, contactless smartcards, open source software, DESFire command set, RFID cryptographic protocols, embedded systems firmware.

That brings me to the most important piece of advice that I can give to all of you: if you've got a good idea, and it's a contribution, I want you to go ahead and DO IT. It is much easier to apologize than it is to get permission. – Grace Hopper

I think a lot of the basis of the open source movement comes from procrastinating students. – Andrew Tridgell

Life would be much easier if I had the source code. – Anonymous

1 Introduction

The Near Field Communication (NFC) protocol is a wireless connectivity standard for short-distance operation over RFID technology. NFC hardware enables

contactless data exchange and authentication between a passive contactless card, or tag (the PICC) and and an active host (the PCD) that is possible over the 13.56 MHz RF band within proximity of approximately 10 centimeters. The NFC protocol is commonly used in applications to open physical NFC-enabled doors, generating reprogrammable spare keys for hotel rooms, renting bicycles or motorized scooters, authenticating access to barriers of restricted parking decks, and charging limited credit transactions via virtual payment card interfaces. Contactless card or tag specifications that are in wide spread use in applictions include the NXP Mifare Classic/Plus/Ultralight series of tags, Mifare DESFire Legacy/EV1/EV2/EV3/Light tags, TI Tag-it branded tags, HID iClass types, LEGIC prime tags, and many other variants of passive contactless cards that are compliant with the ISO-14443 and ISO-15693 standards.

The Chameleon Mini is an open source re-programmable contactless NFC device capable of issuing, receiving and logging bidirectional exchanges of low-level protocol headers and payload data. The recent generation of Chameleon hardware (the RevG series of devices) is frequently employed to analyze the security of RFID based NFC systems. The Chameleon Mini can be leveraged to participate in multiple environment attack scenarios including replay or relay and state restoration attacks, as well as serve as a tool for researchers to sniff and enumerate NFC communication on-the-fly and to perform functional tests of installations of RFID equipment [4]. The freely available firmware sources for the most recent Chameleon Mini profiles are used to emulate as many as eight 8 Kb virtual profiles for contactless cards in a single credit card sized device [2].

The Chameleon RevG hardware offers support for portable and covert on-the-go security analysis of the NFC protocol with a `LIVE` logging feature that prints the recovered data transfers sniffed over its RF antennas over an integrated serial USB interface in realtime. Use of an open source Android OS based scriptable GUI-based logger application developed and actively maintained by the author since 2017 enables quick resetting of emulated NFC tags, sniffing and reading modes as well as the ability to conveniently navigate the Chameleon command terminal over serial USB to actively engage with newly discovered NFC installations [13]. Future generations of NFC sophisticated hardware designed to log, sniff and initiate low-level NFC interactions and exchanges between the PICC and PCD are being developed based on this model platform through the Proxmark3 devices [11].

The proprietary DESFire specification is perhaps the most sophisticated command set and integrated PICC memory storage structure scheme available. It adds an extra layer of security to NFC based data transmissions by including support for modern cryptographic protocols and hash functions [10]. NFC tags that offer DESFire support are common to find in practice, even though they require relatively expensive manufacturing compared to other traditional lighter weight contactless tag technology. Most NFC card specifications feature smaller memory size profiles than the DESFire tags and lack full featured native support for secure modern cryptographic algorithms. DESFire tags are often used to distribute physical university ID cards that secure physical access to resi-

dence locations and buildings and can interact with self-serve vending kiosks to authorize debited transactions based on the exhangeof credentials stored by the contactless IC.

Significance of the project, limitations and related work

We present the progress on NFC emulation software that provides full featured support of the DESFire stack specification for the Chameleon Mini RevG generation of devices. Figure 3 provides a clear illustration of the functional implementation of DESFire emulation on the Chameleon Mini in practice. Support for the DESFire specification is also available using host card emulation (HCE) on Android OS and through PC and MacOS-based exchanges with externally connected NFC hardware using the `LibFreeFare` and `LibNFC` libraries [1, 6, 7]. The work on this project over the last year or so is to the best of our knowledge the first of its kind as a complete and verified functional embedded proof-of-concept implementation that is freely available as open source software to researchers, security experts and end users alike.

Limitations of the current implementation include a lack of interoperability between the Chameleon Mini emulation of these tags and many PCD hosts due to incomplete support for variations of transfer modes involving the different cryptographic integrity checks supported in revisions of the tag specification and default encryption data exchange modes. The reasons for the incomplete support are in part due to the unavailability of recent freely available documentation for the multitude of active revisions of the DESFire tag specifications encountered with in-the-wild applications. Additionally, we surmise that knowledge of the default transfer mode and PCD requirements local to an implementation of a NFC system utilizing contactless DESFire tags is necessary to ensure compatibility and interoperability. That is, a singular expected mode of transfer is non-standard and hard to predict á priori without more knowledge of the working physical environment of the Chameleon Mini emulation support. We have also developed this open source software without the benefit of a substantial toolbox of hardware to support testing nor a large-sized R&D budget on par with that of large manufacturers of this type of NFC technology.

Outline of topics in the article

The next section of the article elaborates on the details of the Chameleon Mini device hardware platform for to which this software project added the DESFire tag NFC support integrated into its open source embedded firmware (see Section 2). An overview of the DESFire tag specification, its unique sophistication and complexity as a modern NFC tag type, and the key features its supports is given in Section 3. The key features of and challenges inherent in writing the open source software contribution documented in this application note are the focus of Section 4. The Appendix A provides a number of annotated examples documenting the low-level byte-wise data exchanges used to communicate with

the Chameleon Mini running the new configuration to natively emulate DESFire tags.

2 The Chameleon Mini device profile and applications

The Chameleon Mini is a programmable standalone tool with a small, portable form factor that facilitates emulation of common NFC tag types, reading, writing and cloning of these tags, and provides integrated support for the sniffing and logging of low-level raw NFC protocol headers and payload data. The Chameleon RevG devices have a few breakthrough features that have distiguished these hardware-based NFC loggers over the last several years. The features of the Chameleon RevG improving on past designs include the following upgrades:

- Integration of a modern AVR XMega chip (the `ATxmega128A4U` [8]) with enough onboard memory space (at 128Kb of FLASH, 8Kb of SRAM, and 2Kb of EEPROM spaces) to provide native emulation modes for over a dozen of passive contactless NFC cards;
- Support for faster FRAM-based memory access;
- Accelerated hardware support for AES and DES cryptographic engines;
- Embedded firmware and flashable bootloader support to memory map the integrated RF hardware on the PCB;
- The ability to upload and download up to 8Kb sized binary memory dumps of NFC tag configurations stored within the onboard FRAM or EEPROM. The data exchanges of the binary dump data are performed using the `XModem` protocol in sequential rounds of 128-byte blocks transferred serially over USB.

The embedded firmware sources and hardware design specification that run on the Chameleon Mini are freely available under an open source license. There is a large international community of developers and users of the Chameleon devices online resulting in frequent modifications to improve on existing support and add new features to the firmware. The integrated serial micro-USB on the integrated AVR XMega chip provides an easy-to-reflash interface over which new firmware binaries (formatted as `avr-gcc` compiler generated `hex` and `eep` files) can be uploaded with the support from the USB bootloader flashed on the device. Several core features provided by the low-level C and assembly language source code for the Chameleon firmware are cited as follows:

- A convenient interface to the Chameleon hardware is provided by a serial terminal that has a human-readable command set. In contrast, traditional mechanisms for communicating with contactless cards over NFC are based on structured APDU-formatted exchanges of hexadecimal data. The Chameleon terminal enables easy on-the-fly reconfiguration of the settings and content stored within as many as eight 8Kb sized partitions of the onboard memory that can be used to emulate the storage of virtualized clones of passive contactless NFC cards;

- The ability to emulate passive NFC devices (e.g., contactless cards and varied tag types);
- The ability to act as an active NFC device (e.g., as a RFID tag reader);
- Support for configuration modes to sniff low-level communication and monitor the raw bidirectional bits transmitted over the RF interface;
- Supported modes to log time-stamped communication details and status events triggered by NFC exchanges to FRAM memory or optionally to print them over the USB interface in realtime via an integrated `LIVE` logging mode. The logging functionality in the RevG devices is designed to archive significant events and data exchanges triggered by any of the emulation, sniffing or reader modes supported in the firmware.

A block diagram illustrating the key modular components of the design of the firmware and the way these components interface with the embedded hardware on the Chameleon PCB is shown in Figure 1.

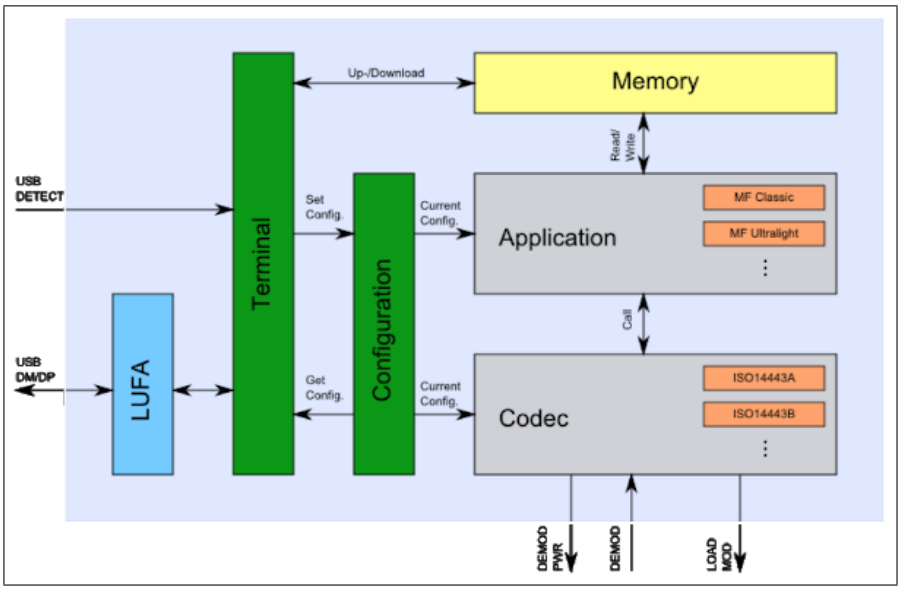

Fig. 1. Block diagram illustrating the core Chameleon Mini firmware design (illustration reproduced from [5] with the permission of David Oswald).

3 The proprietary DESFire tag specification

The Mifare DESFire series of contactless NFC tags conform to a sophisticated and atypically complex proprietary specification over this protocol [10, 9]. NFC

tags of this type are predecessors to the wide spread use of modern secure cryptographic exchange protocols to authenticate short distance NFC based RFID data transfer. As of 2021, there are multiple nested and semi-interoperable generations of DESFire tags, e.g., including the legacy Mifare DESFire, EV1, EV2, EV3 and Light variants. The larger scale integrated memory storage sizes for these tags are usually 2 Kb, 4 Kb or 8Kb [4].

DESFire tags support multiple data transfer modes of command type instructions and data between the PICC and PCD. The format to exchange instructions can either be performed by sending unpadded native commands or by communicating ISO standardized wrapped APDU messages that allow for the PCD to parse both DESFire-specific command sets as well as commands taken from several well known ISO standards [3, 10, 9]. Separate from the format of the messages exchanged over the NFC interface, DESFire tags also support several modes for data tranfer that includes compatibility for both encrypted and unencrypted messages that are optionally padded with crytographically hashed bytes to ensure data integrity over the physical interface using either 2-byte CRC checksums or 4-byte MAC trailers hashed using DES/3DES/AES based primitives.

The precise specification details of the latest generation of DESFire tags are only thinly available to end users from the manufacturer and are as such proprietary in nature. As such, the working details needed to implement an exact clone of these tag types is more difficult than most routes. A fortunate discovery of a publicly posted datasheet for the EV1 generation of DESFire tags from April of 2004 was uncovered by the author a few years ago by data mining for information about these specification details with an internet search engine [10]. The notes and documentation with respect to instruction codes, expected parameters and the conditional return values of the host PCD to the tag upon issuing these commands was indispensible in decoding the secretive operation protocol of NFC tags of this type.

Structure and support for the filesystem in PICC memory Access to PICC memory is flexibly supported and provides a specification to interface with several useful characteristic file types. Individual files are associated to the storage allocations of the physical IC memory into subdirectories called applications that are indexed by a unique 3-byte application identifier (AID). There is a default master (PICC) application with associated master keys for authentication that is the default selected AID upon initial handshaking from PICC to PCD and vice versa. The actively selected AID can be changed via another subsequent structured command call initiated from PICC to PCD.

Within each application space, the file entries are partitioned into data files or records that can store variable length hexadecimal-formatted binary data or signed integer values that can be debited and credited by invoking native instructions. The DESFire specification minimally supports access to the following native file types: standard data files (type 0), backup data files (type 1), value files (type 2), linear record files (type 3), and cyclic record files (type 4).

Each file has 2-bytes of access rights associated with it to indicate one of the following permission categories: read, write, read and write or change. In many instances, the requirements to access sensitive files secured by the cryptographic mechanisms supported by these tags requires both a base round of initial handshaking (PICC-to-PCD) that generates a session key, which is then followed by a crytographic checksum verified exchange of the authentication process using a secret DES/3DES/AES key.

Commands and native instruction support The multiple formats of native and wrapped APDU structured packaged message data over NFC allow for easy transitions between the communication (PICC to PCD and back) of DESFire-only instructions (with `CLA` byte `0x90`), commands compliant with the ISO-14443 and ISO-15693 specifications, or the limited support for a subset of commands taken from the ISO-7816 standard [3]. Figure 2 illustrates the key components of the APDU wrapped data packet exchanges supported by these tags. The ISO-14443(4A) standard provides the protocol and command data formats to initiate and complete the *anti-collision loop* wakeup routines and optional *request for snswer to select* (RATS) queries that are a form of low-level first-contact handshaking between the PICC and PCD. A list of commands that is assimilated from several online references cataloging known DESFire support, their associated 1-byte instruction codes, and high-level descriptions of the functionality provided by each command is listed in Table 1.

CLA	**INS**	**P_1**	**P_2**	**L_c**	**Data Bytes**	**L_e**
`0x90`	command code	`0x00`	`0x00`	variable length of data	command data	`0x00`

Wrapped APDU format for native DESFire commands in the PICC-to-PCD direction (ISO-7816-5 message structure).

Data Bytes	**SW1**	**SW2 (Status)**
DESFire command response data	`0x91`	`0xYY`

Format of the response message for native DESFire commands in the PCD-to-PICC direction. The SW2 status code byte returned by the PCD (denoted by `0xYY` above) is set to either `0x00` to indicate no error in processing the command or is encoded as a reserved byte code to provide an explanation of an error that occurred on the PCD side. The returned error codes are used to indicate problems ranging from hardware errors, to authentication and access permissions errors, to AID and file not found warnings, or to communicate that invalid parameters were passed in the issuing command call.

Fig. 2. Formats of the incoming (PICC to PCD) native-wrapped APDU command structure for the DESFire EV1 tags and the outgoing response format (PCD to PICC) [10].

Command Long Name	INS	Description
AUTHENTICATE	0x0A	Legacy mode authentication
AUTHENTICATE_ISO	0x1A	ISO authentication with 3DES
AUTHENTICATE_AES	0xAA	Standard AES authentication
AUTHENTICATE_EV2_FIRST	0x71	More recent EV2 authentication mode
AUTHENTICATE_EV2_NONFIRST	0x77	More recent EV2 authentication mode
CHANGE_KEY_SETTINGS	0x54	Modify PICC master key properties
SET_CONFIGURATION	0x5C	Used to configure DESFire card or application specific attributes
CHANGE_KEY	0xC4	Changes the key data stored on the PICC
GET_KEY_VERSION	0x64	Returns the active key version stored on the PICC
CREATE_APPLICATION	0xCA	Creates new applications by unique AID
DELETE_APPLICATION	0xDA	Non-restorable deletion operation
GET_APPLICATION_IDS	0x6A	Returns a list of all AID codes stored on the PICC
FREE_MEMORY	0x6E	Returns the total free memory on the tag in bytes
GET_DF_NAMES	0x6D	Obtain the ISO7816-4 DF names associated with the tag
GET_KEY_SETTINGS	0x45	Get permissions data and format for PICC and application master keys
SELECT_APPLICATION	0x5A	Select a specific application by AID for further access
FORMAT_PICC	0xFC	Releases the previously stored user memory (not reversible)
GET_VERSION	0x60	Returns manufacturing header data stored in the PICC
GET_CARD_UID	0x51	Returns the 7-byte card UID assigned by the manufacturer
GET_FILE_IDS	0x6F	Get a list of the file identifiers (by index) within the selected AID
GET_FILE_SETTINGS	0xF5	Obtain properties and permissions about a file
CHANGE_FILE_SETTINGS	0x5F	Modify access permissions of an existing file
CREATE_STDDATA_FILE	0xCD	Add new unformatted binary data storage file type
CREATE_BACKUPDATA_FILE	0xCB	Create unformatted binary file with a shadow backup mechanism
CREATE_VALUE_FILE	0xCC	Create new 32-bit integer storage file
CREATE_LINEAR_RECORD_FILE	0xC1	Create new fixed size file for sequential storage of structurally similar record data structures
CREATE_CYCLIC_RECORD_FILE	0xC0	Similar to the linear record case except that there is a wrap-around storage functionality when the file size limit is exceeded
DELETE_FILE	0xDF	Non-restorable deactivation of a file within the active AID
GET_ISO_FILE_IDS	0x61	Returns a list of the 2-byte file identifiers of all files within the active AID
READ_DATA	0xBD	Read byte-wise contents of standard or backup file types
WRITE_DATA	0x3D	Write data at an offset to stadard or backup file types

GET_VALUE	0x6C	Reads the last permanently stored integer from value records
CREDIT	0x0C	Increase the integer value type in the value type
DEBIT	0xDC	Decrease the integer value type in the value type
LIMITED_CREDIT	0x1C	Increase by a preset limited amount the integer in a value record (must commit the transaction changes at a later time)
WRITE_RECORD	0x3B	Write data to a linear or cyclic record file type
READ_RECORDS	0xBB	List the set of complete records in the associated file type
CLEAR_RECORD_FILE	0xEB	Reset a linear or cyclic record type to an empty state
COMMIT_TRANSACTION	0xC7	Validates the previous write access permissions and credit permissions of all files within the selected AID
ABORT_TRANSACTION	0xA7	Invalidates the previous changes to the files within the selected AID
SELECT	0xA4	ISO7816-4 standard command support
GET_CHALLENGE	0x84	ISO7816-4 standard command support
EXTERNAL_AUTHENTICATE	0x82	ISO7816-4 standard command support
INTERNAL_AUTHENTICATE	0x88	ISO7816-4 standard command support
READ_BINARY	0xB0	ISO7816-4 standard command support
UPDATE_BINARY	0xD6	ISO7816-4 standard command support
READ_RECORDS	0xB2	ISO7816-4 standard command support
APPEND_RECORD	0xE2	ISO7816-4 standard command support

Table 1. Listing of the DESFire command set integrated into the new Chameleon Mini firmware emulation support [10].

4 Key features of and challenges writing the embedded software implementation

The majority of testing and development for the project to add support for DESFire tag emulation was done with the open source `LibNFC` library in C starting from stock firmware sources compiled using the `avr-gcc` compiler toolchain on MacOS and Linux. The results of the author's work on this project was merged into the main firmware sources [2] in October of 2020 after approximately six to eight months of active development[1]. The following crucial modifications and extensions of the application layer, codec layer and logging support modules (refer to Figure 1) in the code were necessary to finalize fully featured Chameleon DESFire support:

- New native AES support using hardware acceleration support from the integrated Microchip AVR XMega microcontrollers;
- Extensions of prior work to add hardware based DES and 3DES support to the Chameleon Mini firmware;

[1] See https://github.com/emsec/ChameleonMini/pull/287.

- A built-in customized extension of the Chameleon terminal commands to enhance DESFire configuration support for users (See Figure 3);
- Enhancements to the `LIVE` logging functionality of the Chameleon RevG devices. The implementation of this new "tick" functionality added to the live logging required a non-standard linked list implementation whereby we handled with the AVR memory structures by assigning pointer values with calls to the C standard library function `memcpy(3)`.

The modifieed DESFire firmware has been confirmed as non-functional on the RevE generation of Chameleon Mini devices due to memory constraints and the lack of hardware acceleration for cryptographic functions by the integrated AVR microcontroller (the `ATxmega32A4U` chip).

During the development process (from start to finish) there were a number of road blocks to overcome. The significant obstacles encountered in writing the native DESFire firmware support for the Chameleon Mini include the next issues:

- We were forced by local embedded system constraints to cleverly optimize and organize our use of the embedded AVR memory to resolve insufficient memory type exceptions throughout the development process. Most notably, the structures and buffer space needed to store cryptographic structures for use with AES and 3DES were carefully leveraged on the stack to avoid unrecoverable overflow errors and race conditions.
- After first trying to resolve support for cryptographic primitives purely at the software level by adapating open source numerical libraries for AVR chips, tests conclusively identified the need for the hardware accelerated cryptographic engines developed by Microchip. As noted in [4], the speedup in computations for AES and 3DES operations provides an order of magnitude improvement available within the expected short timing windows for data responses over NFC from PICC to PCD.
- A complicated nested, quasi-linked pointer based structure was required to efficiently store the filesystem entries and tag accounting metadata (such as unique AID sequences, keys and key versions). This structure requires sequential access to arrays within memory that store further pointers to locations in memory to interface with and coordinate access to the distinct EEPROM/FRAM/FLASH sections on the integrated AVR XMega chip.

The runtime behavior of our new Chameleon DESFire tag emulation was carefully tested and debugged with `LibNFC` [7] on MacOS in conjunction with an external USB-powered NFC reader that interfaced with the Chameleon device connected to a logging interface running the *Chameleon Mini Live Debugger* GUI-based application on Android OS [12, 13]. Documentation to verify correctnessserves as a benchmark reference point for what future revisions of the source code should generate. The `LibNFC` based testing code developed produced the examples of wrapped DESFire command exchanges cited for reference in Appendix A (see page 13).

The resulting emulation support provides not only an interface for the DESFire command set, but also critically allows security researchers to clone the unique maufacturer header data that is presumed as a sacred marker by which a PCD gateway point can uniquely identify any user in the NFC system. These sensitive, hard-to-clone data fields include the tag ATS bytes, hardware and software version bytes, batch number, production date bytes, and the 7-byte UID assigned by the manufacturer to uniquely distinguish NFC based passive PICC devices when they are manufactured in production. The sublistings shown in Figure 3 provide a working example of basic configuration and handshaking with an emulated DESFire tag configuration on the Chameleon Mini by the end user.

5 Acknowlegements, credits and funding sources

The sources for the DESFire emulation support in the public firmware sources for the Chameleon Mini software and embedded firmware are derived from a fork of the main project [2] due to Dmitry Janushkevich in 2017. The author credits Professor Josephine Yu and the School of Mathematics at the Georgia Institute of Technology for allowing me to work on this as a secondary project as a Ph.D. candidate over the Summer and Fall of 2020 and for providing funding for hardware used in testing of the project. We similarly thank the Kasper and Oswald (KAOS) manufacturers of the original generations of Chameleon Mini devices for providing support in the form of discounted devices to support our efforts on the project development.

References

1. Android HCE DESFire: A software implementation of Desfire in an Android app. https://github.com/jekkos/android-hce-desfire
2. Chameleon Mini Firmware (authoritative sources). https://github.com/emsec/ChameleonMini
3. ISO/IEC 14443, 15693 and 7816 Standards. Identification Cards - Contactless Integrated Circuit Cards. www.iso.org
4. Kasper T., von Maurich I., Oswald D., Paar C. (2011) Chameleon: A Versatile Emulator for Contactless Smartcards. In: Rhee KH., Nyang D. (eds) Information Security and Cryptology - ICISC 2010. ICISC 2010. Lecture Notes in Computer Science, vol 6829. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-24209-0_13
5. Kasper, T. and Oswald, D. Presentation slides on the history of the Chameleon Mini devices. https://raw.github.com/wiki/emsec/ChameleonMini/Images/160110_ChameleonMini_history_smaller.pdf
6. LibFreeFare: A convenience API for NFC cards manipulations on top of LibNFC. https://github.com/nfc-tools/libfreefare
7. LibNFC: A platform independent NFC library. https://github.com/nfc-tools/libnfc
8. Microchip. ATxmega1284U Data Sheet. https://ww1.microchip.com/downloads/en/DeviceDoc/ATxmega128-64-32-16A4U-DataSheet-DS40002166A.pdf
9. NXP Semiconductors. MIFARE DESFire Functional specification. Publicly available MF3ICD81 datasheet (2008). https://tinyurl.com/kwweanp9

```
> CONFIG=MF_DESFIRE
> DF_SETHDR=ATS 0675F7B102
> UID=2377000B99BF98
```

Chameleon Mini terminal input for firmware compiled with DESFire emulation support to configure an IBM-JCOP branded NFC tag.

```
NFC reader: SCM Micro / SCL3711-NFC&RW opened

Sent bits:     26 (7 bits)
Received bits: 03  44
Sent bits:     93  20
Received bits: 88  23  77  00  dc
Sent bits:     93  70  88  23  77  00  dc  4b  b3
Received bits: 04
Sent bits:     95  20
Received bits: 0b  99  bf  98  b5
Sent bits:     95  70  0b  99  bf  98  b5  2f  24
Received bits: 20
Sent bits:     e0  50  bc  a5
Received bits: 75  77  81  02  80
Sent bits:     50  00  57  cd

Found tag with
 UID: 2377000b99bf98
ATQA: 4403
 SAK: 20
 ATS: 75  77  81  02  80
```

Output of reading the resulting DESFire tag emulated by the Chameleon Mini device using an externally connected USB NFC reader with the LibNFC `nfc-anticol` *utility [7].*

```
[usb] pm3 --> hf 14a read
[+]  UID: 23 77 00 0B 99 BF 98
[+] ATQA: 44 03
[+]  SAK: 20 [1]
[+]  ATS: 75 77 81 02 80
[=] field dropped.
```

Confirmed support of the firmware emulation support using the Proxmark3 hardware.

Fig. 3. Verification of DESFire emulation support using cloned manufacturer data for an IBM-JCOP tag with a randomized 7-byte UID.

10. Philips Semicondictors. Mifare DESFire: Contactless multi-application IC with DES and 3DES security. Publicly available MF3-IC-D40 datasheet (2004). https://tinyurl.com/5era3dx2

11. Proxmark III. A Radio Frequency IDentification Tool. http://www.proxmark.org

12. Schmidt, M. D. Chameleon Mini DESFire Stack (development sources). https://github.com/maxieds/ChameleonMiniDESFireStack
13. Schmidt, M. D. Chameleon Mini Live Debugger. https://github.com/maxieds/ChameleonMiniLiveDebugger

A Examples of the emulated DESFire tag authentication and wrapped APDU command exchanges

A.1 Testing the DESFire application management command set

```
>>> Select Application By AID:
    -> 90 5a 00 00 03 00 00 00 | 00
    <- 91 00
>>> Start AES Authenticate:
    -> 90 aa 00 00 01 00 00
    <- 54 b8 9e fe 19 9b c6 a5 | fd 8f 00 be c1 23 99 c0 | 91 af
    -> 90 af 00 00 10 df a0 79 | 13 59 ac 4c 75 5f 81 69 |
       bc 9c 3e c6 7e 00
    <- a9 e2 79 42 11 63 9c 14 | 07 b3 02 2f 2e 4b 2e c5 | 91 00
>>> Get AID List From Device:
    -> 90 6a 00 00 00 00
    <- 77 88 99 01 00 34 91 00
>>> CreateApplication command:
    -> 90 ca 00 00 05 77 88 99 | 0f 03 00
    <- 91 de
>>> Get AID List From Device:
    -> 90 6a 00 00 00 00
    <- 77 88 99 01 00 34 91 00
>>> CreateApplication command:
    -> 90 ca 00 00 05 01 00 34 | 0f 03 00
    <- 91 de
>>> Get AID List From Device:
    -> 90 6a 00 00 00 00
    <- 77 88 99 01 00 34 91 00
>>> CreateApplication command:
    -> 90 ca 00 00 05 77 88 99 | 0f 03 00
    <- 91 de
>>> Get AID List From Device:
    -> 90 6a 00 00 00 00
    <- 77 88 99 01 00 34 91 00
>>> Select Application By AID:
    -> 90 5a 00 00 03 77 88 99 | 00
    <- 91 00
>>> Start AES Authenticate:
    -> 90 aa 00 00 01 00 00
    <- f4 1a ea 8f ab cd a2 89 | c9 78 e4 da 99 7c 0a 98 | 91 af
    -> 90 af 00 00 10 49 04 ee | b6 bc 9b 22 63 7e 0e 19 |
       ea cb 7f 1c e5 00
    <- 26 71 50 b0 a8 01 da 95 | 15 90 23 e3 0c de 6e b9 | 91 00
```

```
>>> DeleteApplication command:
    -> 90 da 00 00 03 77 88 99 | 00
    <- 91 00
>>> Get AID List From Device:
    -> 90 6a 00 00 00 00
    <- 01 00 34 91 00
>>> Select Application By AID:
    -> 90 5a 00 00 03 00 00 00 | 00
    <- 91 00
>>> Start AES Authenticate:
    -> 90 aa 00 00 01 00 00
    <- d2 92 d5 f7 81 de 5f e2 | 8d 32 38 1a b2 44 74 88 | 91 af
    -> 90 af 00 00 10 2b d2 cf | 9f 86 e5 76 ac 30 50 45 |
       ac ec 0c 4e ea 00
    <- ff 8d 5a c6 3c cc 9f 01 | 9e b4 06 8d 09 5d cb ca | 91 00
>>> DeleteApplication command:
    -> 90 da 00 00 03 01 00 34 | 00
    <- 91 00
>>> Get AID List From Device:
    -> 90 6a 00 00 00 00
    <- 91 00
>>> DeleteApplication command:
    -> 90 da 00 00 03 00 00 00 | 00
    <- 91 9d
>>> Get AID List From Device:
    -> 90 6a 00 00 00 00
    <- 91 00
```

A.2 Using the DESFire key management commands

```
>>> Select Application By AID:
    -> 90 5a 00 00 03 00 00 00 | 00
    <- 91 00
>>> Start AES Authenticate:
    -> 90 aa 00 00 01 00 00
    <- 1f 97 25 a3 ee 58 29 04 | f1 b9 d3 da fa 61 a9 2d | 91 af
    -> 90 af 00 00 10 ca 3a df | 84 8e 24 5d 8e 1a 3d 61 |
       3f 81 25 2b 62 00
    <- ca f1 0f 36 8f a5 34 31 | 87 b5 fc dd 2c 7d 1a b7 | 91 00
>>> ChangeKey command:
    -> 90 c4 00 00 11 00 00 00 | 00 00 00 00 00 00 00 00 |
       00 00 00 00 00 00 00 00
    <- 91 00
>>> Select Application By AID:
    -> 90 5a 00 00 03 00 00 00 | 00
    <- 91 00
>>> Start AES Authenticate:
    -> 90 aa 00 00 01 00 00
    <- ea 32 f1 97 06 1d d8 99 | ed 6b bc ca 71 d6 d5 13 | 91 af
```

```
    -> 90 af 00 00 10 3f 4b 03 | 28 cc 22 9b cc 8a 5e 32 |
       9e b9 44 4a 60 00
    <- 4d 5f da 4e 47 7a 58 a2 | 6a 6f b3 69 2d f0 8b ea | 91 00
>>> GetKeySettings command:
    -> 90 45 00 00 00 00
    <- 0f 03 91 00
>>> ChangeKeySettings command:
    -> 90 54 00 00 01 0f 00
    <- 91 00
>>> GetKeyVersion command:
    -> 90 64 00 00 01 00 00
    <- 02 91 00
```

A.3 Using the DESFire file management commands

```
>>> CreateApplication command:
    -> 90 ca 00 00 05 01 00 34 | 0f 03 00
    <- 91 00
>>> Select Application By AID:
    -> 90 5a 00 00 03 01 00 34 | 00
    <- 91 00
>>> Start AES Authenticate:
    -> 90 aa 00 00 01 00 00
    <- c8 d6 4e ee 3a d2 7c 1b | 44 71 76 62 b6 ec 9d 0a | 91 af
    -> 90 af 00 00 10 a8 d0 f8 | ae 30 64 ae 23 e5 20 18 |
       38 e5 1d 1d 33 00
    <- a5 22 c2 3b 5f f2 bc 9e | 18 0e 87 c0 71 ca ef 69 | 91 00
>>> CreateStdDataFile command:
    -> 90 cd 00 00 07 00 00 0f | 00 04 00 00 00
    <- 91 00
>>> CreateBackupDataFile command:
    -> 90 cb 00 00 07 01 00 0f | 00 08 00 00 00
    <- 91 00
>>> CreateValueFile command:
    -> 90 cc 00 00 11 02 00 0f | 00 00 00 00 00 00 01 00 |
       00 80 00 00 00 01 00
    <- 91 00
>>> CreateLinearRecordFile command:
    -> 90 c1 00 00 0a 03 00 0f | 00 04 00 00 0c 00 00 00
    <- 91 00
>>> CreateCyclicRecordFile command:
    -> 90 c0 00 00 0a 04 00 0f | 00 01 00 00 05 00 00 00
    <- 91 00
>>> GetFileIds command:
    -> 90 6f 00 00 00 00
    <- 00 01 02 03 04 91 00
>>> DeleteFile command:
    -> 90 df 00 00 01 01 00
    <- 91 00
```

```
>>> GetFileIds command:
    -> 90 6f 00 00 00 00
    <- 00 02 03 04 91 00
>>> GetFileSettings command:
    -> 90 f5 00 00 01 00 00
    <- 00 00 0f 00 04 00 00 91 | 00
>>> GetFileSettings command:
    -> 90 f5 00 00 01 02 00
    <- 02 00 0f 00 00 00 00 00 | 00 01 00 00 80 00 00 00 | 01 91 00
>>> GetFileSettings command:
    -> 90 f5 00 00 01 03 00
    <- 03 00 0f 00 04 00 00 0c | 00 00 00 00 00 91 00
>>> GetFileSettings command:
    -> 90 f5 00 00 01 04 00
    <- 04 00 0f 00 01 00 00 05 | 00 00 00 00 00 91 00
```

A.4 Using the DESFire file manipulation commands

```
>>> CreateStdDataFile command:
    -> 90 cd 00 00 07 00 00 0f | 00 04 00 00 00
    <- 91 de
>>> CreateBackupDataFile command:
    -> 90 cb 00 00 07 01 00 0f | 00 08 00 00 00
    <- 91 de
>>> GetFileIds command:
    -> 90 6f 00 00 00 00
    <- 00 01 91 00
>>> ReadData command:
    -> 90 bd 00 00 07 00 00 00 | 00 04 00 00 00
    <- 8f f9 2d 9f 91 00
>>> DeleteFile command:
    -> 90 df 00 00 01 01 00
    <- 91 00
>>> GetFileIds command:
    -> 90 6f 00 00 00 00
    <- 00 91 00
>>> ReadData command:
    -> 90 bd 00 00 07 00 00 00 | 00 04 00 00 00
    <- 8f f9 2d 9f 91 00
>>> ReadData command:
    -> 90 bd 00 00 07 00 02 00 | 00 02 00 00 00
    <- 8f f9 91 00
>>> WriteData command:
    -> 90 3d 00 00 0b 00 00 00 | 00 04 00 00 00 00 00 00 | 00
    <- 91 00
>>> ReadData command:
    -> 90 bd 00 00 07 00 00 00 | 00 04 00 00 00
    <- 00 00 00 00 91 00
>>> WriteData command:
```

```
    -> 90 3d 00 00 09 00 02 00 | 00 02 00 00 04 05 00
    <- 91 00
>>> ReadData command:
    -> 90 bd 00 00 07 00 00 00 | 00 04 00 00 00
    <- 8f 09 04 05 91 00
>>> CreateValueFile command:
    -> 90 cc 00 00 11 02 00 0f | 00 00 00 00 00 00 01 00 |
       00 80 00 00 00 01 00
    <- 91 de
>>> GetFileIds command:
    -> 90 6f 00 00 00 00
    <- 02 91 00
>>> GetValue command:
    -> 90 6c 00 00 01 02 00
    <- a0 00 00 00 91 00
>>> Credit(ValueFile) command:
    -> 90 0c 00 00 05 02 40 00 | 00 00 00
    <- 91 00
>>> GetValue command:
    -> 90 6c 00 00 01 02 00
    <- a0 00 00 00 91 00
>>> Debit(ValueFile) command:
    -> 90 dc 00 00 05 02 40 00 | 00 00 00
    <- 91 00
>>> GetValue command:
    -> 90 6c 00 00 01 02 00
    <- a0 00 00 00 91 00
>>> LimitedCredit(ValueFile) command:
    -> 90 1c 00 00 05 02 20 00 | 00 00 00
    <- 91 00
>>> GetValue command:
    -> 90 6c 00 00 01 02 00
    <- a0 00 00 00 91 00
>>> CommitTransaction command:
    -> 90 c7 00 00 00 00
    <- 91 00
>>> GetValue command:
    -> 90 6c 00 00 01 02 00
    <- c0 00 00 00 91 00
>>> Debit(ValueFile) command:
    -> 90 dc 00 00 05 02 c8 00 | 00 00 00
    <- 91 be
>>> CommitTransaction command:
    -> 90 c7 00 00 00 00
    <- 91 00
>>> GetValue command:
    -> 90 6c 00 00 01 02 00
    <- c0 00 00 00 91 00
>>> CreateValueFile command:
    -> 90 cc 00 00 11 03 00 0f | 00 00 00 00 00 00 01 00 |
```

```
       00 80 00 00 00 00 00
    <- 91 00
>>> GetFileIds command:
    -> 90 6f 00 00 00 00
    <- 02 03 91 00
>>> GetValue command:
    -> 90 6c 00 00 01 03 00
    <- 80 00 00 00 91 00
>>> Credit(ValueFile) command:
    -> 90 0c 00 00 05 03 40 00 | 00 00 00
    <- 91 00
>>> GetValue command:
    -> 90 6c 00 00 01 03 00
    <- 80 00 00 00 91 00
>>> Debit(ValueFile) command:
    -> 90 dc 00 00 05 03 40 00 | 00 00 00
    <- 91 00
>>> GetValue command:
    -> 90 6c 00 00 01 03 00
    <- 80 00 00 00 91 00
>>> LimitedCredit(ValueFile) command:
    -> 90 1c 00 00 05 03 20 00 | 00 00 00
    <- 91 9d
>>> GetValue command:
    -> 90 6c 00 00 01 03 00
    <- 80 00 00 00 91 00
>>> AbortTransaction command:
    -> 90 a7 00 00 00 00
    <- 91 00
>>> GetValue command:
    -> 90 6c 00 00 01 03 00
    <- 80 00 00 00 91 00
>>> CreateLinearRecordFile command:
    -> 90 c1 00 00 0a 01 00 0f | 00 02 00 00 06 00 00 00
    <- 91 de
>>> GetFileIds command:
    -> 90 6f 00 00 00 00
    <- 01 91 00
>>> ReadRecords command:
    -> 90 bb 00 00 07 01 00 00 | 00 06 00 00 00
    <- 00 00 00 e5 b9 b2 91 00
>>> WriteRecords command:
    -> 90 3b 00 00 0a 01 00 00 | 00 03 00 00 00 00 00 00
    <- 91 00
>>> ReadRecords command:
    -> 90 bb 00 00 07 01 00 00 | 00 06 00 00 00
    <- 00 00 00 e5 b9 b2 91 00
```

A.5 Testing the DESFire general utility command set

```
>>> GetVersion command:
    -> 90 60 00 00 00 00
    <- 04 01 01 00 01 18 05 91 | af
    -> 90 af 00 00 00 00
    <- 04 01 01 00 01 18 05 91 | af
    -> 90 af 00 00 00 00
    <- 3f 0f 20 82 5e eb 16 ff | ff ff ff ff ff ff 91 00
>>> FormatPICC command:
    -> 90 fc 00 00 00 00
    <- 91 00
>>> GetCardUID command:
    -> 90 51 00 00 00 00
    <- 3f 0f 20 82 5e eb 16 91 | 00
>>> FreeMemory command:
    -> 90 6e 00 00 00 00
    <- 60 00 91 0c
```