IMPORTANT

- This workshop requires a laptop with Linux (VM or host)
- If you have not done so yet, install the proxmark3 client:

```
wget https://totallylegit.net/orange24.tar.gz
tar xf orange24.tar.gz
cd orange24
./build_proxmark3.sh
```

Getting familiar with DESFire

Sebastiaan Groot

OrangeCon 05-09-2024

$ whoami

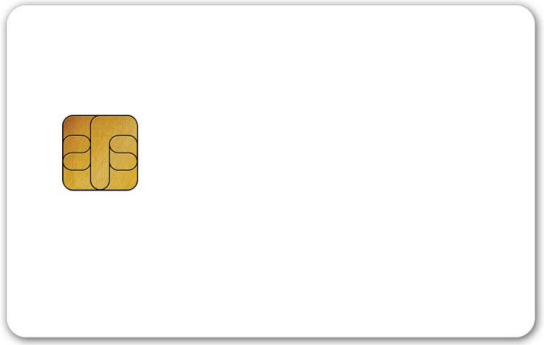
- Sebastiaan Groot
- Ethical Hacker @ KPN
- Binary exploitation / System Security
- Love for DFIR & CTFs

$ cat desfire/intro

- Made by NXP as part of their Mifare line
- Built on top of ISO-14443 Type A (14a)
  - Same standard Mifare Classic, NTAG 215 and more uses
- Most technical docs under NDA
  - but plenty of reversing has been done
- Uses 13.56MHz as frequency
- Allows many “applications” per card
- Not (yet) cryptographically broken when used correctly

$ cat desfire/terminology

- PICC: Proximity Integrated Circuit Card (the card)
- PCD: Proximity Coupling Device (the reader)



$ cat 14a/intro

The 14a standard defines:

- Physical characteristics
- Radio specs
- Initialization and anticollision protocol
- Transmission protocol

Useful to dive into docs if debugging something in pre-DESFire, 14a traffic

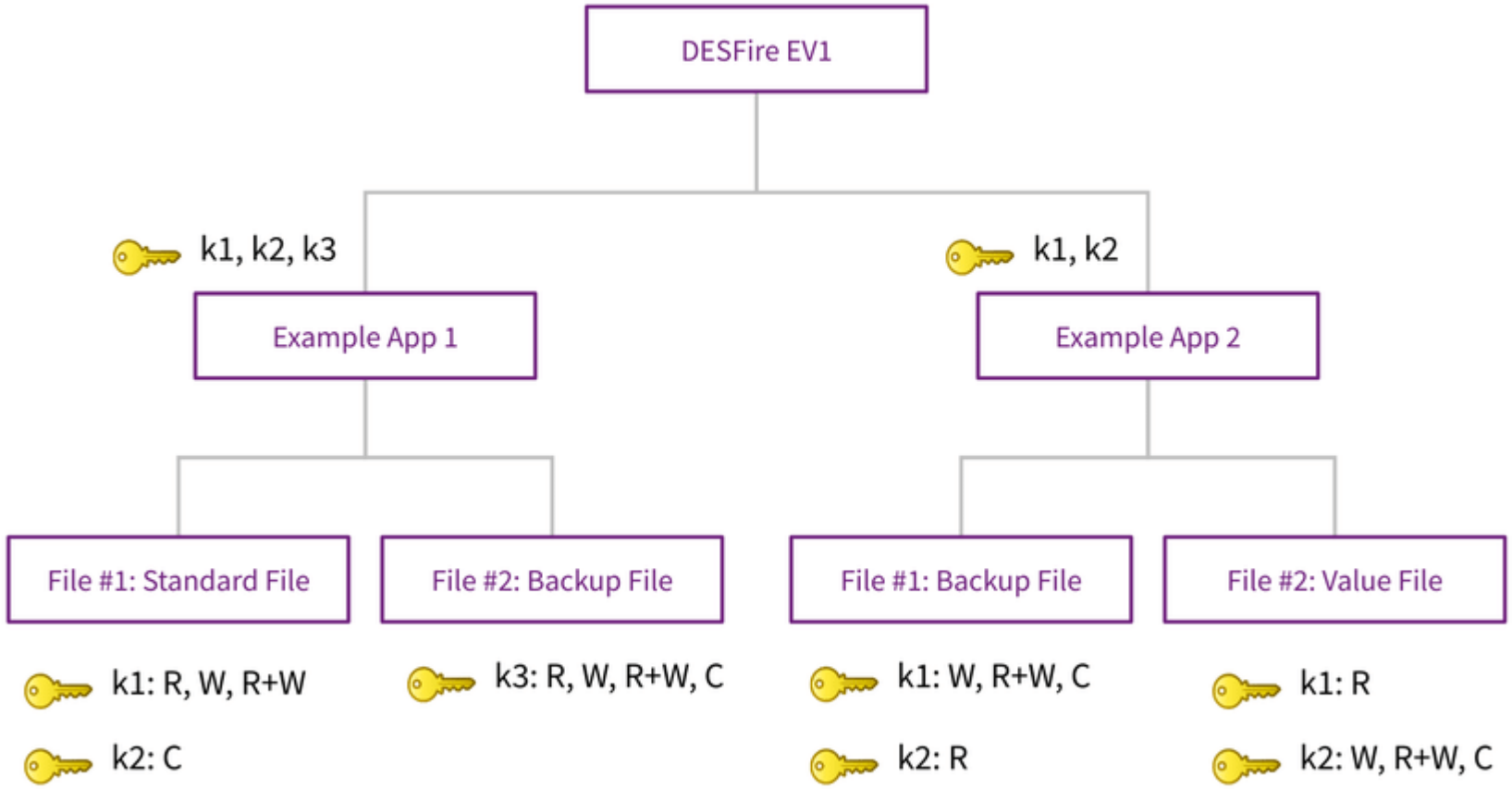
$ feh 14a/card-selection.png

Start	End	Src	Data (! denotes parity error)	CRC	Annotation
0	1056	Rdr	26(7)		REQA
2228	4596	Tag	44 03		
13072	15536	Rdr	93 20		ANTICOLL
16708	22532	Tag	88 04 62 21 cf		
31248	41712	Rdr	93 70 88 04 62 21 cf e0 cf	ok	SELECT_UID
42948	46468	Tag	24 d8 36	ok	
54992	57456	Rdr	95 20		ANTICOLL-2
58628	64452	Tag	6a e1 6e 80 65		
73360	83824	Rdr	95 70 6a e1 6e 80 65 4a 69	ok	SELECT_UID-2
85060	88644	Tag	20 fc 70	ok	
95504	100272	Rdr	50 00 57 cd	ok	HALT
248672	249664	Rdr	52(7)		WUPA
250900	253268	Tag	44 03		
261664	272128	Rdr	93 70 88 04 62 21 cf e0 cf	ok	SELECT_UID
273364	276884	Tag	24 d8 36	ok	
283936	294400	Rdr	95 70 6a e1 6e 80 65 4a 69	ok	SELECT_UID-2
295620	299204	Tag	20 fc 70	ok	
309088	313856	Rdr	e0 81 b8 62	ok	RATS - FSDI=8, CID=1
315028	324308	Tag	06 75 77 81 02 80 02 f0	ok	

$ cat desfire/basics

- Cards have applications
- Applications have keys, files and settings
  - Just fancy encrypted directories
- Master application 0 has master key for card-global operations

$ feh desfire/structure.png



$ cat desfire/applications

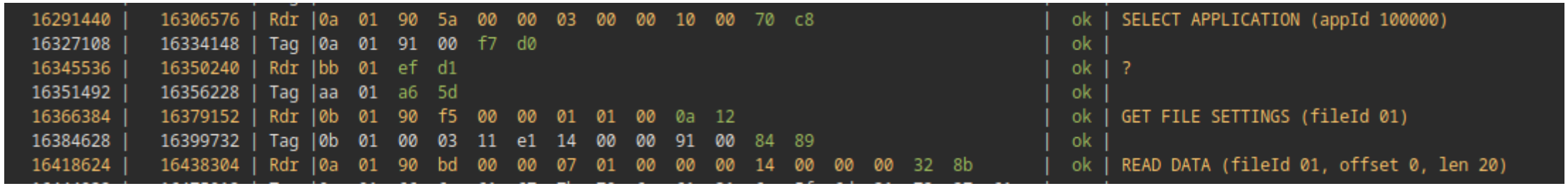
- Each application has own set of keys and files
- Creating or modifying applications might be possible without auth
  - Depends on global settings

- Some known application ids:
  - pm3_src/client/resources/aid_desfire.json

$ cat desfire/crypto

- Each application has 1 cipher configured
  - Choices: DES, 2TDEA, 3TDEA, AES128
- 1 to 14 keys per application
- Keys are used to derive session keys
- Encrypted data sent using CBC-mode of chosen cipher

$ feh desfire/select_application.png



```
16291440 |   16306576 | Rdr |0a  01  90  5a  00  00  03  00  00  10  00  70  c8                  |  ok | SELECT APPLICATION (appId 100000)
16327108 |   16334148 | Tag |0a  01  91  00  f7  d0                                              |  ok |
16345536 |   16350240 | Rdr |bb  01  ef  d1                                                      |  ok | ?
16351492 |   16356228 | Tag |aa  01  a6  5d                                                      |  ok |
16366384 |   16379152 | Rdr |0b  01  90  f5  00  00  01  01  00  0a  12                          |  ok | GET FILE SETTINGS (fileId 01)
16384628 |   16399732 | Tag |0b  01  00  03  11  e1  14  00  00  91  00  84  89                  |  ok |
16418624 |   16438304 | Rdr |0a  01  90  bd  00  00  07  01  00  00  00  14  00  00  00  32  8b  |  ok | READ DATA (fileId 01, offset 0, len 20)
```

$ cat desfire/files

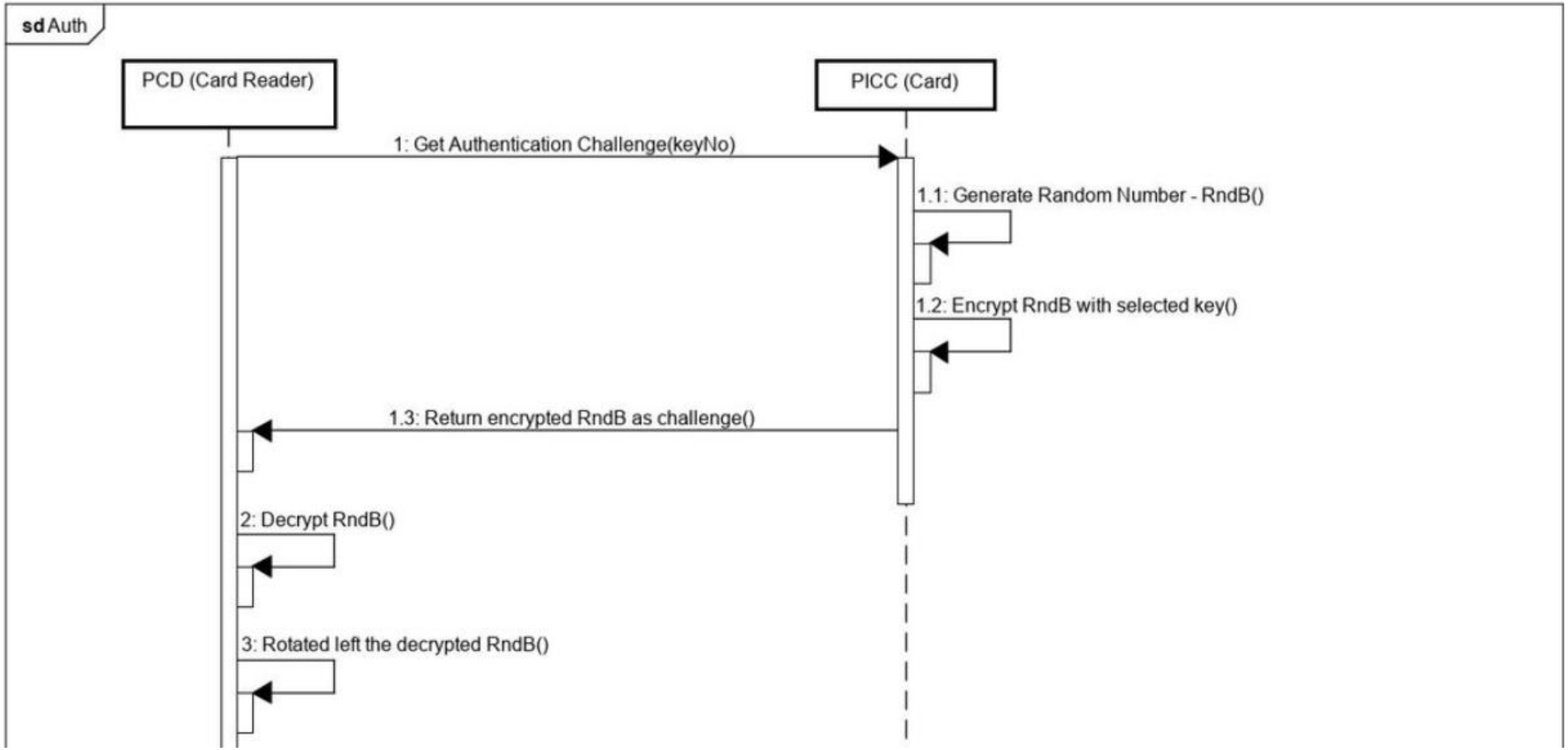
- 6 types of files:
  - Standard, backup, value, cyclic record, linear record, transaction mac
- 3 communication modes:
  - Plain, mac, encrypted
- 4 access right types:
  - Read / write / read-write / change
  - Each can be set to key0 to keyD, free access or deny access

$ cat desfire/auth

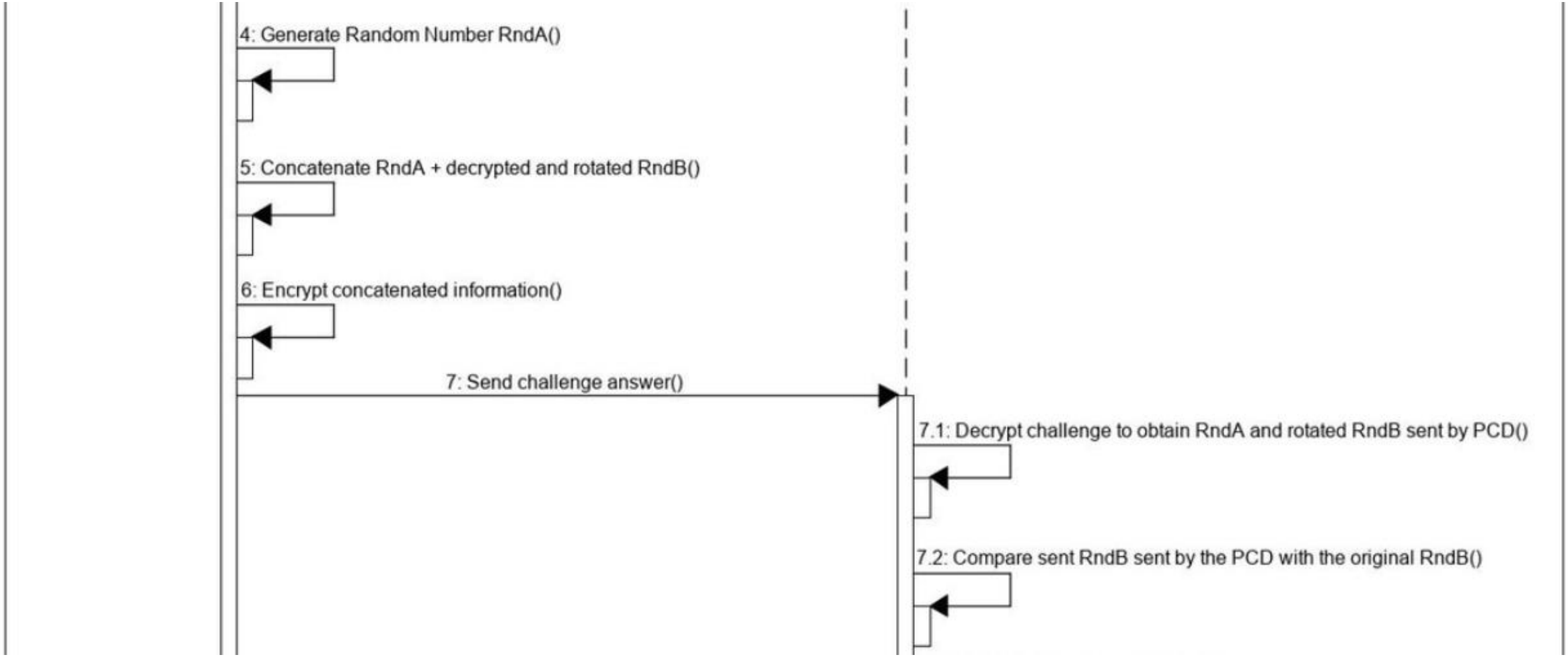
Uses a 3-pass authentication protocol

- Reader and card have pre-shared key
- Reader and card generate randoms RndA (reader) and RndB (card)
- Session key is concatenation of part of RndA and RndB

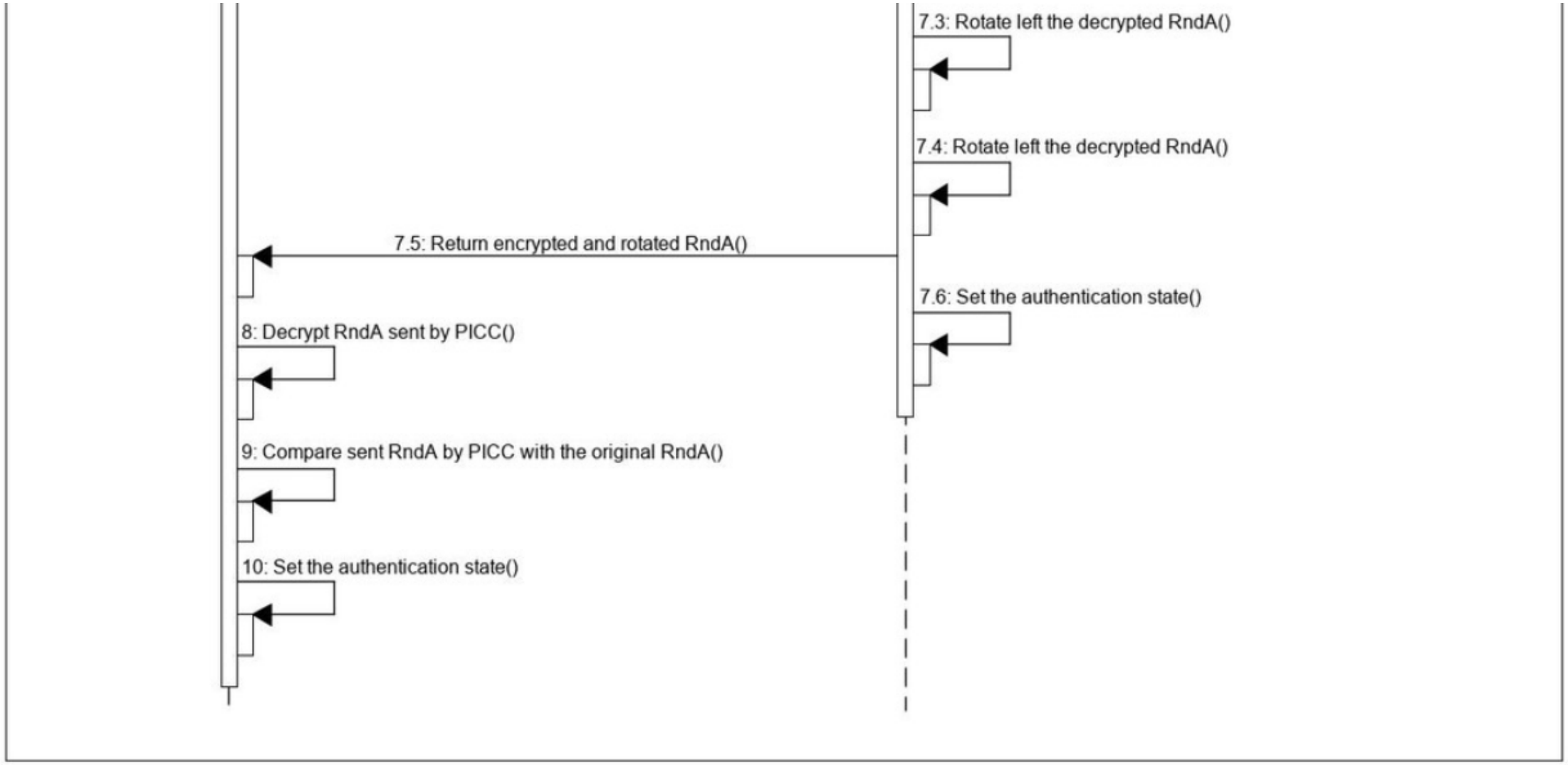
$ feh desfire/auth1.png



$ feh desfire/auth2.png



$ feh desfire/auth3.png



$ cat pm3/programming_desfire

- Proxmark3 has good support for interacting with DESFire cards
- Quick walkthrough through following operations:
  - Sniffing
  - App operations
  - Files operations
  - Key operations

$ cat pm3/setup

- On Linux:
  - sed -e ‘s/PM3RDV4/PM3GENERIC/’ Makefile.platform.sample > Makefile.platform
  - make clean && make -j
  - sudo ./pm3 # if your user is not allowed to directly access USB serial

$ feh pm3/desfire_sniffing1.png

- Many ways to position card, pm3 and reader - try to see what works

$ feh pm3/desfire_sniffing2.png

```
[usb] pm3 --> hf 14a sniff
[=] Press pm3 button to abort sniffing

[#] trace len = 764
[usb] pm3 --> hf mfdes list
[+] Recorded activity ( 764 bytes )
[=] start = start of start frame. end = end of frame. src = source of transfer.
[=] ISO14443A - all times are in carrier periods (1/13.56MHz)

     Start |       End | Src | Data (! denotes parity error)                                  | CRC | Annotation
-----------+-----------+-----+----------------------------------------------------------------+-----+-------------------
         0 |      1056 | Rdr |26                                                              |     | REQA
      2228 |      4596 | Tag |44  03                                                          |     |
     13072 |     15536 | Rdr |93  20                                                          |     | ANTICOLL
     16708 |     22532 | Tag |88  04  62  21  cf                                              |     |
     31248 |     41712 | Rdr |93  70  88  04  62  21  cf  e0  cf                              |  ok | SELECT_UID
     42948 |     46468 | Tag |24  d8  36                                                      |  ok |
```

$ feh pm3/desfire_apps1.png

```
[usb] pm3 --> hf mfdes lsapp

[+] ---------------------------------- PICC level ----------------------------------
[+] Applications count: 5 free memory 2560 bytes
[+] PICC level auth commands:
[+]    Auth.............. YES
[+]    Auth ISO......... YES
[+]    Auth AES......... NO
[+]    Auth Ev2......... NO
[+]    Auth ISO Native... YES
[+]    Auth LRP......... NO
[+] PICC level rights:
[+] [1...] CMK Configuration changeable   : YES
[+] [.1..] CMK required for create/delete : NO
[+] [..1.] Directory list access with CMK : NO
[+] [...1] CMK is changeable              : YES
[+]
[+] Key: 2TDEA
[+] key count: 1
[+] PICC key 0 version: 0 (0x00)

[+] ------------------------------ Applications list ------------------------------
[+] Application number: 0x123456
[+]   ISO id.... 0x0000
[+]   DF name....  ( 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 )
```

$ feh pm3/desfire_apps2.png

```
[usb] pm3 --> hf mfdes createapp --aid 111111 --dstalgo AES
[+] Desfire application 111111 successfully created
```

```
[usb] pm3 --> hf mfdes deleteapp --aid 111111 -t AES
[+] Desfire application 111111 deleted
```

```
[usb] pm3 --> hf mfdes auth -n 0 -t 2tdea -k 00000000000000000000000000000000 --aid 111111
[+] Application AID 111111 selected and authenticated succesfully
[+] Context:
[=] Key num: 0 Key algo: 2tdea Key[16]: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[=] Secure channel: ev1 Command set: niso Communication mode: plain
[=] Session key [16]: 01 02 03 04 82 56 76 3B 01 02 03 04 82 56 76 3B
[=]     IV [8]: 00 00 00 00 00 00 00 00
```

$ feh pm3/desfire_files1.png

```
[usb] pm3 --> hf mfdes createfile --aid 111111 --fid 01 --size 000010
[=] ---- Create file settings ----
[+] File type         : Standard data
[+] File number       : 0x01 (1)
[+] File ISO number   : n/a
[+] File comm mode    : Plain
[+] Additional access: No
[+] Access rights     : EEEE
[+]   read......... free
[+]   write........ free
[+]   read/write... free
[+]   change....... free
[=] File size (bytes)... 16 / 0x10
[+] Standard data file 01 in the app 111111 created successfully
```

$ feh pm3/desfire_files2.png

```
[usb] pm3 --> hf mfdes chfilesettings --aid 111111 --fid 01 --rrights free --wrights key0 --rwrights key0 --chrights key0 --no-auth
[=] ---- Set file settings ----
[+] File comm mode   : Plain
[+] Additional access: No
[+] Access rights    : e000
[+]   read......... free
[+]   write........ key 0x00
[+]   read/write... key 0x00
[+]   change....... key 0x00
[+] File settings changed successfully
```

$ feh pm3/desfire_files3.png

```
[usb] pm3 --> hf mfdes write --aid 111111 --fid 01 -d deadbeef
[=] Write data file 01 success
[usb] pm3 --> hf mfdes read --aid 111111 --fid 01
[=] ------------------------------ File 01 data ------------------------------
[+] Read 16 bytes from file 0x01 offset 0
[=]  Offset  | Data                                                | Ascii
[=] ----------------------------------------------------------------------------
[=]   0/0x00 | DE AD BE EF 00 00 00 00 00 00 00 00 00 00 00 00 | ................
```

$ feh pm3/desfire_keys.png

```
[usb] pm3 --> hf mfdes changekey --aid 111111 --newkeyno 1 --newkey FFFFFFFFFFFFFFFF
[=] Changing key for AID 111111
[=] auth key 0: des [8] 00 00 00 00 00 00 00 00
[=] changing key number  0x01 (1)
[=] old key: des [8] 00 00 00 00 00 00 00 00
[=] new key: des [8] FF FF FF FF FF FF FF FF
[+] Change key ( ok )
```

$ feh pm3/load_trace.png

```
[segrt@segrt:...enarios/training/challenges]$ pm3 --offline
[=] Session log /home/segrt/.proxmark3/logs/log_20240415094011.txt
[+] loaded `/home/segrt/.proxmark3/preferences.json`
[=] OFFLINE mode. Check "proxmark3 -h" if it's not what you want.



  8888888b.  888b     d888  .d8888b.
  888   Y88b 8888b   d8888 d88P  Y88b
  888    888 88888b.d88888      .d88P
  888   d88P 888Y88888P888     8888"
  8888888P"  888 Y888P 888      "Y8b.
  888        888  Y8P  888 888    888
  888        888   "   888 Y88b  d88P
  888        888       888  "Y8888P"    [ ◓  ]


  [ Proxmark3 RFID instrument ]


[offline] pm3 --> trace load -f chal1.trace
[+] loaded 537 bytes from binary file `chal1.trace`
[+] Recorded Activity (TraceLen = 537 bytes)
[?] try `trace list -1 -t ...` to view trace.  Remember the `-1` param
[offline] pm3 --> hf mfdes list
[-] ⛔ You requested a trace upload in offline mode, consider using parameter `-1` for working from Tracebuffer
[+] Recorded activity ( 537 bytes )
[=] start = start of start frame. end = end of frame. src = source of transfer.
[=] ISO14443A - all times are in carrier periods (1/13.56MHz)

     Start |        End | Src | Data (! denotes parity error)                                                  | CRC | Annotation
-----------+------------+-----+--------------------------------------------------------------------------------+-----+----------------
         0 |       1056 | Rdr |26                                                                              |     | REQA
      2228 |       4596 | Tag |44  03                                                                          |     |
```

$ cat pm3/outro

- hf mfdes help -> list of all pm3 desfire commands
- Try -h for all commands, lots of options and examples

$ cat chals

- 3 challenges to solve
- Each has sniffed traces between real card and reader
- Your goal is to “clone” each of the three cards

$ cat info

- Find this presentation and the challenges here:

https://totallylegit.net/orange24.tar.gz

Challenge 1: Top of the line security

The security team bought these new fancy DESFire cards and quickly implemented a secret access code on the cards. The readers are programmed to read the secret access code from the cards and only allow access when the returned value is correct. The trace below was sniffed between a working card and the reader.

Hint: The file content is in the format: `flag{.*}`, but to beat the challenge you must get the reader to light up the top LED with your own card.

Files

- `chal1.trace`

Commands

```
# analysis
trace load -f ../challenges/1-top-of-the-line-security/chal1.trace
hf mfdes list

# programming
hf mfdes createapp --aid <app-id>
hf mfdes createfile --aid <app-id> --fid <file-id>
hf mfdes write --aid <app-id> --fid <file-id> -d <hex-encoded file contents>
```

$ challenges/demo_1.sh

Challenge 2: Encrypted comms

Having learned from their mistakes, the security team made sure the secret access file is now marked to use an encrypted channel for file transfer. The trace below was sniffed between a working card and the reader, but the file contents look encrypted. You spot an employee in the lobby with an access card within antenna distance.

Hint: The file content is in the format: `flag{.*}`, but to beat the challenge you must get the reader to light up the middle LED with your own card.

Files

- `chal2.trace`
- Physical Chal2 DESFire card (ask Sebastiaan)

Commands

```
# analysis - step 1
trace load -f ../challenges/2-encrypted-comms/chal2.trace
hf mfdes list

# analysis - step 2
hf mfdes lsapp
hf mfdes lsfiles --aid <app-id>
hf mfdes read --aid <app-id> --fid <file-id> --no-auth

# programming
hf mfdes createapp --aid <app-id>
hf mfdes createfile --aid <app-id> --fid <file-id>
hf mfdes write --aid <app-id> --fid <file-id> -d <hex-encoded file contents>
```

Challenge 3: Inner sanctum

You have managed to gain access to the building, but the server room is still off-limits! Turns out the admin of this room created a separate application with stricter settings for access. They even managed to hand-roll different encryption keys for each unique card. Good luck getting into this one!

At least you have managed to capture two valid traces from one working card and the reader.

Hint: The file content is in the format: `[\x00]*flag{.*}`, but to beat the challenge you must get the reader to light up the bottom LED with your own card.

Files

- `chal3-1.trace`
- `chal3-2.trace`

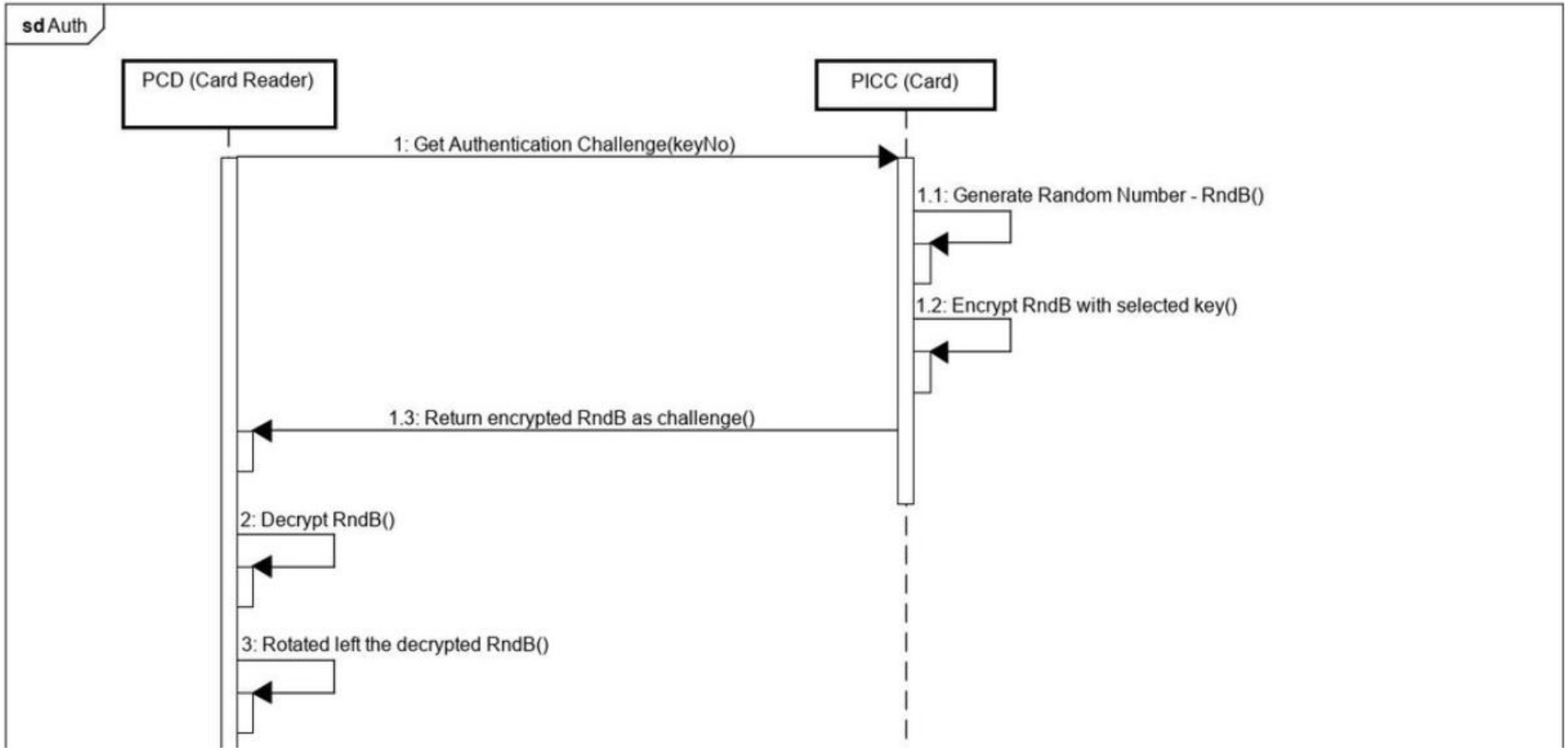
Commands

```
# programming
hf mfdes createapp --aid <app-id> --dstalgo <app cipher>
hf mfdes changekey --newalgo <app cipher> --newkeyno <key number> --newkey <hex-encoded key>
hf mfdes createfile --aid <app-id> --fid <file-id>
hf mfdes write --aid <app-id> --fid <file-id> -d <hex-encoded file contents>
```

Hints

- This challenge is very error-prone.
  - There are two trace files just to allow you to more easily identify which parts of the messages you should extract.
  - Which parts of the messages are different?
  - How many bytes would you expect the encrypted RndA / RndB / file contents be?
- Use the prepended null bytes in the file to your advantage!
  - In order to cleanly decrypt the full plaintext, you would need to keep the IV state fully correct when you start decrypting the file contents.
  - Since the file contents are prepended with known bytes, you can afford the first AES-CBC block to be mangled during decryption.
- The chosen key scheme is the following:
  - `(9*\x00 bytes) .. (7-byte UID)`
- The python3 script skeleton `decrypt.py` can be used to recreate the session key and decrypt the file from the trace
  - This script requires the `pycryptodome` package

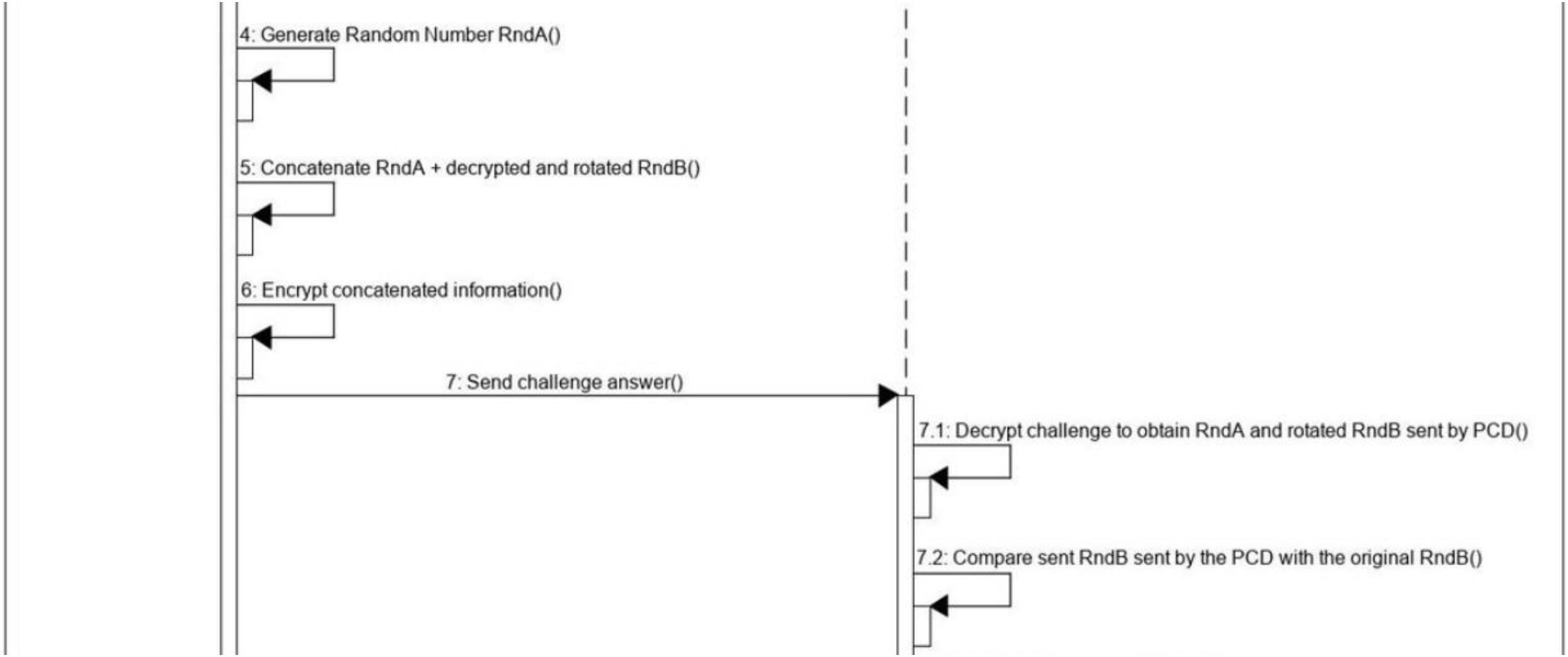
$ feh desfire/auth1.png



```
11637744 |   11652880 | Rdr |0A  01  90  5A  00  00  03  02  00  10  00  06  F1                  |  ok | SELECT APPLICATION (appI
11675684 |   11682724 | Tag |0A  01  91  00  F7  D0                                              |  ok |
11696400 |   11709168 | Rdr |0B  01  90  AA  00  00  01  01  00  E2  6F                          |  ok | AUTH AES (keyNo 1)
11746116 |   11771588 | Tag |0B  01  A2  E3  1C  D2  21  87  16  95  EB  C5  39  14  6C  04  5F  2F |     |
         |            |     |91  AF  F3  B5                                                      |  ok |
11810912 |   11859456 | Rdr |0A  01  90  AF  00  00  20  57  21  FE  58  46  6A  BA  87  22  26  0F |     |
         |            |     |8C  7D  00  EF  16  53  37  6E  6A  12  4D  21  7E  6C  8E  DD  FA  04 |     |
         |            |     |30  E9  DE  00  E7  4C                                              |  ok | AUTH FRAME / NEXT FRAME
11896596 |   11922004 | Tag |0A  01  2E  AA  19  36  A2  36  31  1C  37  50  56  5F  5B  1E  51  41 |     |
         |            |     |91  00  FD  6A                                                      |  ok |
```

```python
#!/usr/bin/env python3

import binascii
from Crypto.Cipher import AES

# Message & key material
msg1 = binascii.unhexlify('A2E31CD221871695EBC539146C045F2F')
msg2 = binascii.unhexlify('5721FE58466ABA8722260F8C7D00EF1653376E6A124D217E6C8EDDFA0430E9DE')
key =  9 * b'\x00' + binascii.unhexlify('0462216AE16E80')
iv  = 16 * b'\x00'
```

$ feh desfire/auth2.png



```
11637744 |   11652880 | Rdr |0A  01  90  5A  00  00  03  02  00  10  00  06  F1                  |  ok | SELECT APPLICATION (appId 100
11675684 |   11682724 | Tag |0A  01  91  00  F7  D0                                              |  ok |
11696400 |   11709168 | Rdr |0B  01  90  AA  00  00  01  01  00  E2  6F                          |  ok | AUTH AES (keyNo 1)
11746116 |   11771588 | Tag |0B  01  A2  E3  1C  D2  21  87  16  95  EB  C5  39  14  6C  04  5F  2F  |     |
         |            |     |91  AF  F3  B5                                                      |  ok |
11810912 |   11859456 | Rdr |0A  01  90  AF  00  00  20  57  21  FE  58  46  6A  BA  87  22  26  0F  |     |
         |            |     |8C  7D  00  EF  16  53  37  6E  6A  12  4D  21  7E  6C  8E  DD  FA  04  |     |
         |            |     |30  E9  DE  00  E7  4C                                              |  ok | AUTH FRAME / NEXT FRAME
11896596 |   11922004 | Tag |0A  01  2E  AA  19  36  A2  36  31  1C  37  50  56  5F  5B  1E  51  41  |     |
         |            |     |91  00  FD  6A                                                      |  ok |
```

```python
#!/usr/bin/env python3

import binascii
from Crypto.Cipher import AES

# Message & key material
msg1 = binascii.unhexlify('A2E31CD221871695EBC539146C045F2F')
msg2 = binascii.unhexlify('5721FE58466ABA8722260F8C7D00EF1653376E6A124D217E6C8EDDFA0430E9DE')
key =  9 * b'\x00' + binascii.unhexlify('0462216AE16E80')
iv  = 16 * b'\x00'
```

Hints

1. This challenge is very error-prone. There are two trace files just to allow you to more easily identify which parts of the mes
   parts of the messages are different? How many bytes would you expect the encrypted RndA / RndB / file contents be?

2. Use the prepended null bytes in the file to your advantage! In order to cleanly decrypt the full plaintext, you would need to
   you start decrypting the file contents. Since the file contents are prepended with known bytes, you can afford the first AES-
   decryption.

3. The chosen key scheme is the following: `(9*\x00 bytes) .. (7-byte UID)`

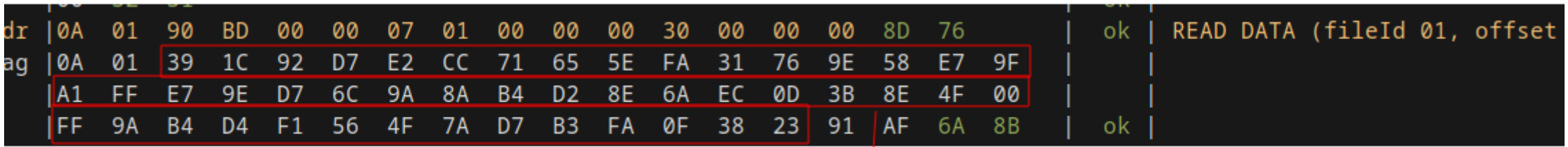
4. The following python3 skeleton can be used to recreate the session key (requires the `pycryptodome` package):

```
#!/usr/bin/env python3
```

```
    13072 |      15536 | Rdr |93  20                                                    |     | ANTICOLL
    16724 |      22548 | Tag |88  04  62  21  CF                                       |     |
    31248 |      41712 | Rdr |93  70  88  04  62  21  CF  E0  CF                       |  ok | SELECT_UID
    42964 |      46484 | Tag |24  D8  36                                                |  ok |
    55312 |      57776 | Rdr |95  20                                                    |     | ANTICOLL-2
    58948 |      64772 | Tag |6A  E1  6E  80  65                                        |     |
    73680 |      84144 | Rdr |95  70  6A  E1  6E  80  65  4A  69                        |  ok | SELECT_UID-2
```

```
#!/usr/bin/env python3

import binascii
from Crypto.Cipher import AES

# Message & key material
msg1 = binascii.unhexlify('A2E31CD221871695EBC539146C045F2F')
msg2 = binascii.unhexlify('5721FE58466ABA8722260F8C7D00EF1653376E6A124D217E6C8EDDFA0430E9DE')
key =  9 * b'\x00' + binascii.unhexlify('0462216AE16E80')
iv  = 16 * b'\x00'
```

```
# Recover RndB from msg1
cipher = AES.new(key, AES.MODE_CBC, iv=iv)
RndB = cipher.decrypt(msg1)

# Recover RndA from msg2
tmp = cipher.decrypt(msg2)
RndA = tmp[:16]

# Construct session key
sesskey = RndA[:4] + RndB[:4] + RndA[12:] + RndB[12:]
```




```
# Decrypt encrypted file contents
ciphertext = binascii.unhexlify('391C92D7E2CC71655EFA31769E58E79FA1F
FE79ED76C9A8AB4D28E6AEC0D3B8E4F00FF9AB4D4F1564F7AD7B3FA0F3823')
cipher = AES.new(sesskey, AES.MODE_CBC, iv=iv)
plaintext = cipher.decrypt(ciphertext)
```

```
[segrt@segrt:.../challenges/3-inner-sanctum]$ python3 decrypt-trace1.py
AES input:
msg1: b'a2e31cd221871695ebc539146c045f2f'
msg2: b'5721fe58466aba8722260f8c7d00ef1653376e6a124d217e6c8eddfa0430e9de'
key: b'00000000000000000000462216ae16e80'
iv: b'00000000000000000000000000000000'
RndA: b'42dd37cf783d227636e7a6c6fd5c742b'
RndB: b'c9e5d61af7dcad354d98bfec34acf076'
SessKey: b'42dd37cfc9e5d61afd5c742b34acf076'
Ciphertext: b'391c92d7e2cc71655efa31769e58e79fa1ffe79ed76c9a8ab4d28e6aec0d3b8e4f00ff9ab4d4f1564f7ad7b3fa0f3823'
Plaintext: b"\xff'L&\xdb-X\x939j\x08B\\\xae\x95E\x00\x00\x00\x00\x00\x00\x00\x00flag{d3crypt_fr0m_tr4c3}"
```