NFC
libnfc & nfc-tools

Course SPVC2012

Tech Talk

Sebastian Büttrich, pITLab
sebastian@itu.dk

Scope

- Requirements: what we already know
- Types of NFC cards/tags
- Data structures - words, blocks, sectors
- Hardware: NFC readers/writers
- Reading and writing NFC tags with libnfc, nfc-tools, ACS SDK tools
- Example: sketch of a simple *'pitcoin'*

Requirements: what we already know

General understanding of RFID, NFC, radio waves

Power and data transfer

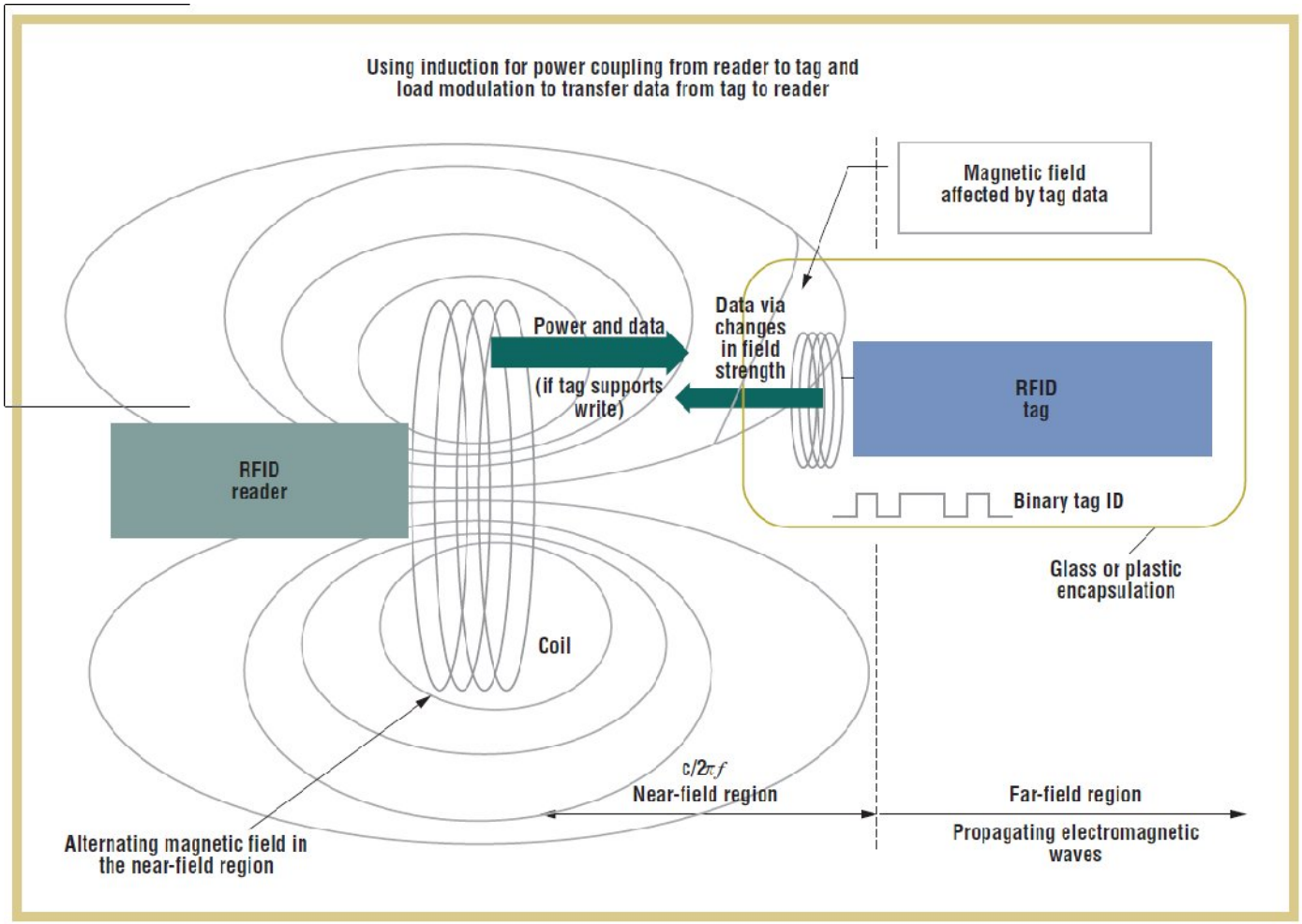


Figure 3. Near-field power/communication mechanism for RFID tags operating at less than 100 MHz.

Want, R.; , "An introduction to RFID technology," Pervasive Computing, IEEE , vol.5, no.1, pp. 25- 33, Jan.-March 2006.

Requirements: what we already know

NFC standard

compatible with RFID ISO 15693 and 14443

compatible with the FeliCa and Mifare smart card standards

Data rate 424 kbps

Operating in the 13.56-MHz band (22 meter)

Types of NFC tags/cards

ISO 14443: Mifare

Jewel

FeliCa

MIFARE

NXP Semiconductors-owned trademark

Seven different kinds of contactless cards

MIFARE

MIFARE Classic

employ a proprietary protocol compliant to parts (but not all) of ISO/IEC 14443-3 Type A , with an NXP proprietary security protocol for authentication and ciphering.

MIFARE Ultralight

low-cost ICs that employ the same protocol as MIFARE Classic, but without the security part and slightly different commands

MIFARE Ultralight C

the first low-cost ICs for limited-use applications that offer the benefits of an open Triple DES cryptography

MIFARE DESFire
are smart cards that comply to ISO/IEC 14443-4 Type A with a mask-ROM operating system from NXP.
MIFARE DESFire EV1
includes AES encryption.

MIFARE Plus
drop-in replacement for MIFARE Classic with certified security level (AES 128 based)

MIFARE SAM AV2
secure access module that provides the secure storage of cryptographic keys and cryptographic functions

MIFARE Classic 1k

http://www.mifare.net/products/mifare-smartcard-ic-s/mifare-1k/

1 kbyte EEPROM (768 Byte free available)
Unique serial number UID (4 Byte and 7 Byte)
16 securely separated sectors supporting multi-application
Each sector consists 4 blocks with a length of 16 Byte
2 x 48 bit keys per sector for key hierarchy
Access conditions free configurable based on 2 keys
Number of single write operations: 100.000
Data retention: 10 years

MIFARE Classic 1k

- Most popular low price card in e.g. Transportation, Ticketing, ID, etc
- Used a.o. In London Oyster, DK Rejsekort, ITU ID card
- Infamous for its flawed security model:
  weak cipher (3 booleans on 4 bits), non-random random generator, keys not independent
- (Nohl et al., 2008; de Koning Gans et al., 2008)

MIFARE transaction in detail

Courtois, Nicolas T.

THE DARK SIDE OF SECURITY BY OBSCURITY

and Cloning MiFare Classic Rail and Building Passes, Anywhere, Anytime

http://eprint.iacr.org/2009/137.pdf

Typical transaction flow in MiFare Classic, following (de Koning Gans et al., 2008; Garcia et al., 2008) and using the same notations:

1. First the reader and the card engage in the anticollision protocol where the reader learns the unique ID of the card and selects the card.

2. The reader issues a command ’60 XX’ or ’61 XX’ by which it starts the mutual symmetric-key authentication process between the card and the reader, with the key pertaining to the block number XX.
3. The card answers with a random nT on 4 bytes,
4. The reader sends a cryptogram on 8 bytes which is fnRg = nR©ks1 and faRg = suc2(nT)©ks2.
5. The card responds with 4 bytes, suc3(nT)©ks3.
6. Then all subsequent communications and data are encrypted and the card will now accept read, write and increment commands for block XX.
Here nR is the 32-bit nonce chosen by the reader,

fnRg is the encryption of it, suc is a certain bijective function, and (ks1;ks2;ks3) are the 96 bits of the keystream produced by the Crypto-1 stream cipher after being initialized with nT and nR. We refer to (de Koning Gans et al., 2008) for more details.

Data structures

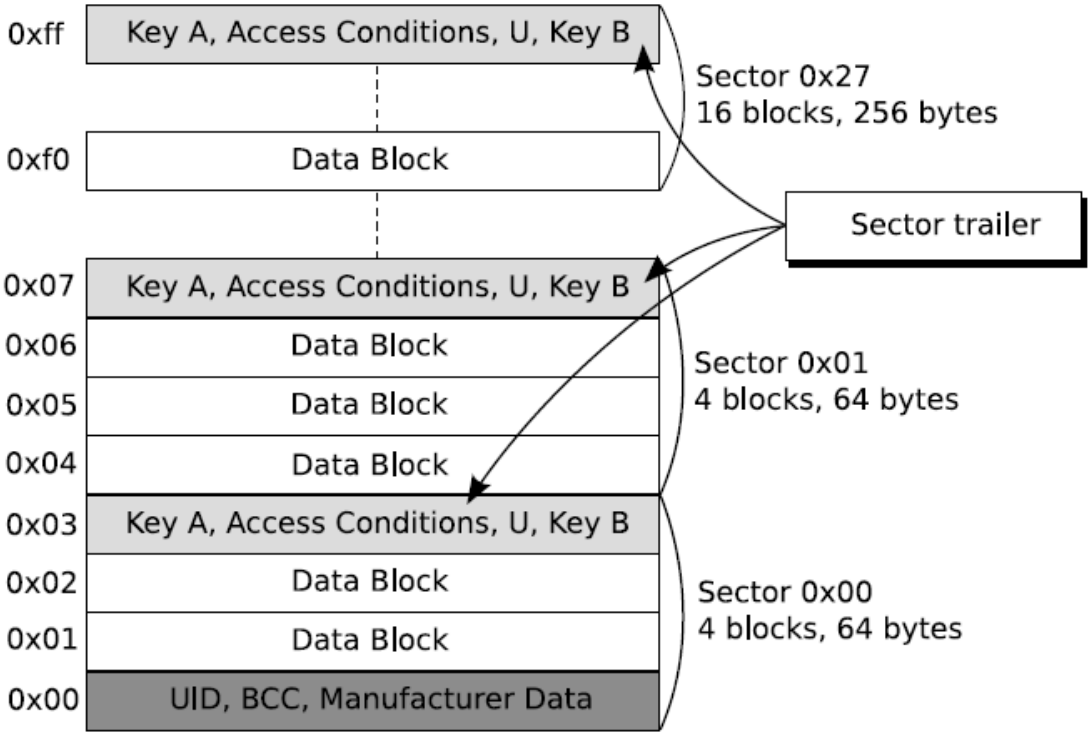


Fig. 1. Logical structure

Data structure

64 blocks of 16 bytes
16 sectors of 4 blocks

2 hex digits make 1 byte

Word = 4 hex digits
Block = 8 words
Every block has a key, every sector has keys a, b and instructions for their use

First sectors typically hold UID, vendor info

BLOCK

SECTOR

```
    OFFSET
    V

00000000:  0a 88 30 1c ae 08 04 00  62 63 64 65 66 67 68 69  ..0.....bcdefghi  <<< BLOCK
00000010:  14 01 03 e1 03 e1 03 e1  03 e1 03 e1 03 e1 03 e1  ................
00000020:  03 e1 03 e1 03 e1 03 e1  03 e1 03 e1 03 e1 03 e1  ................
00000030:  a0 a1 a2 a3 a4 a5 78 77  88 c1 ff ff ff ff ff ff  ......xw........ |
00000040:  00 00 03 13 d1 01 0f 54  02 65 6e 6a 75 73 74 20  .......T.enjust  |
00000050:  61 20 74 65 73 74 0a fe  00 00 00 00 00 00 00 00  a test.......... | <<< SECTOR
00000060:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................ |
00000070:  d3 f7 d3 f7 d3 f7 7f 07  88 40 ff ff ff ff ff ff  .........@......
```

Hardware: Readers/Writers

Hardware: libnfc compatibility

Dongle

Manufacturer	Product	NFC Controller	Host bus	Depends	Driver	Tested	Support	Additionnal note
SCM Microsystems	SCL3710	PN531 v4.2	USB	libusb	PN53X_USB	YES	YES	
	SCL3711	PN533 v2.7	USB	libusb	PN53X_USB	YES	YES	
Sensor ID	Stick ID	PN533 v2.7	USB	libusb	PN53X_USB	YES	YES	
TOPPAN FORMS	TN31CUD001SW	PN531 v4.2	USB	libusb	PN53X_USB	YES	YES	
Snapper	Feeder	PN531 v3.4	USB	libusb	PN53X_USB	YES	YES	
T-money	T-Pop	?	USB	libusb	-	NO	N/A[1]	

Flat

Manufacturer	Product	NFC Controller	Host bus	Depends	Driver	Tested	Support	Additionnal note
ASK	LoGO	PN533 v2.7	USB	libusb	PN53X_USB	YES	LIMITED	This device can not act as target[1]
ACS	ACR122U101	PN532 v1.4	USB	PCSC	ACR122	YES	LIMITED	This device can be only be used as initiator (not target nor P2P)[2]
	ACR122U206	PN532 v1.4	USB	PCSC	ACR122	YES	LIMITED	This device can be only be used as initiator (not target nor P2P)[2]
tikitag	ACR122U102	PN532 v1.4	USB	PCSC	ACR122	YES	LIMITED	This device can be only be used as initiator (not target nor P2P)[2]
touchatag	ACR122U102	PN532 v1.4	USB	PCSC	ACR122	YES	LIMITED	This device can be only be used as initiator (not target nor P2P)[2]
ARYGON Technologies	ADRA-USB	PN531 v?	USB	-	ARYGON	YES	YES	
	ADRB-USB	PN532 v1.4	USB	-	ARYGON	YES	YES	PN532 is connected to SiLabs UART-USB bridge

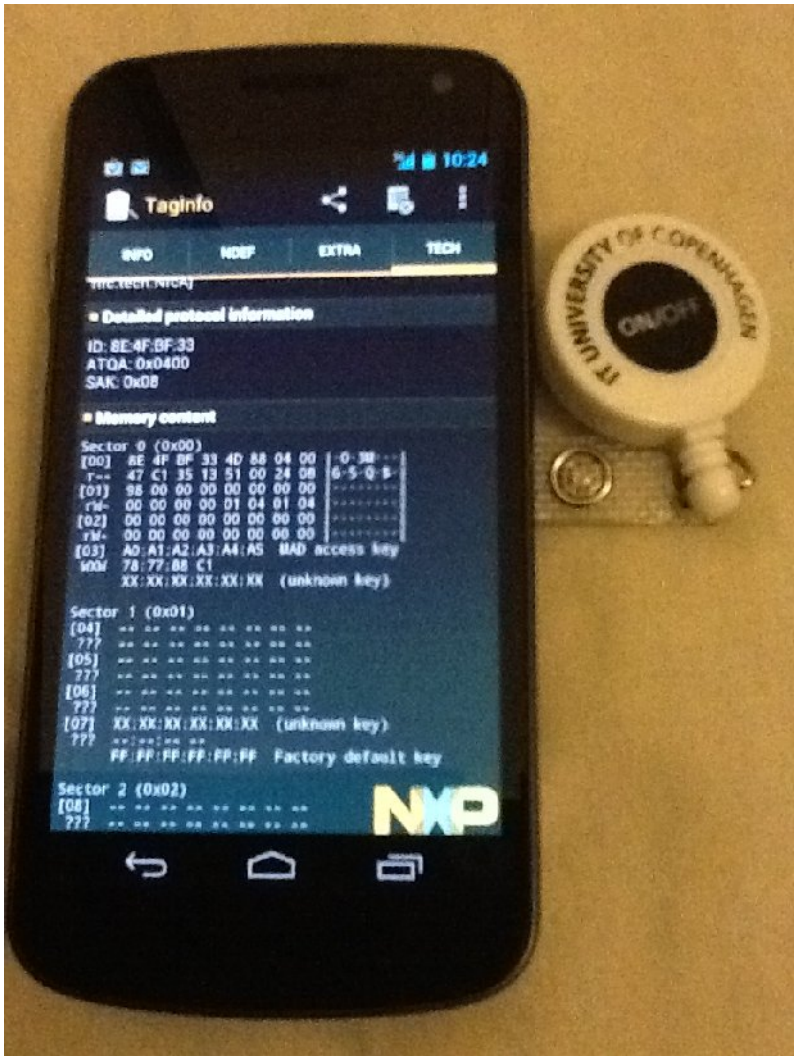
Hardware: mobile phones

Nokia, Samsung, HTC, LG, and many more

http://www.nfcworld.com/nfc-phones-list/#available

Perspective: Mobile payment, Wallet

Hardware: mobile phones



Hardware: Tagstand Kit

= ACS ACR122U

Software

Requirements:
summary: http://www.itu.dk/pit/?n=Main.NFCInfo

- Drivers (pcsc for all OSs)
  http://www.acs.com.hk/index.php?pid=drivers&id=ACR122U

- libnfc http://www.libnfc.org/
- nfc-tools http://code.google.com/p/nfc-tools/
  and/or
- ACS SDK

- Interfaces to libnfc exist for various environments/languages

  Useful:

- hex editor (e..g hexer, ghex2, ..)

- compare tools for files - diff, meld, ...

Note: examples here are based on GNU/Linux (Ubuntu), however the principles are the same regardless of OS

Installation and test

Use ACR122U, empty cards/tags.

Install drivers, test:

```
pcsc_scan
```

Install libnfc & nfc-tools, test:

```
nfc-list -v
```

pcsc_scan should result in something like:

```
# pcsc_scan
PC/SC device scanner
...
Scanning present readers...
0: ACS ACR122U 00 00
..
Thu Mar 22 13:17:27 2012
 Reader 0: ACS ACR122U 00 00
  Card state: Card inserted,
  ATR: 3B 8F 80 01 80 4F 0C A0 00 00 03 06 03 00 02 00 00 00 00 69

ATR: 3B 8F 80 01 80 4F 0C A0 00 00 03 06 03 00 02 00 00 00 00 69

+ TS = 3B --> Direct Convention
+ T0 = 8F, Y(1): 1000, K: 15 (historical bytes)
  TD(1) = 80 --> Y(i+1) = 1000, Protocol T = 0
-----
  TD(2) = 01 --> Y(i+1) = 0000, Protocol T = 1
```

```
-----
+ Historical bytes: 80 4F 0C A0 00 00 03 06 03 00 02 00 00 00 00
  Category indicator byte: 80 (compact TLV data object)
    Tag: 4, len: F (initial access data)
      Initial access data: 0C A0 00 00 03 06 03 00 02 00 00 00 00
+ TCK = 69 (correct checksum)

Possibly identified card (using /usr/share/pcsc/smartcard_list.txt):
3B 8F 80 01 80 4F 0C A0 00 00 03 06 03 00 02 00 00 00 00 69
   Mifare card with 4k EEPROM
```

smartcards_list, excerpt:

```
3B 8E 80 01 10 38 77 A7 80 91 E1 65 D0 00 42 00 00 82 72
    Czech Republic e-Passport (issued Feb 2009)

3B 8E 80 01 80 91 91 31 C0 64 77 E3 03 00 83 82 90 00 1C
    Belgian Passport

3B 8F 01 80 25 A0 00 00 00 56 57 44 4B 34 30 30 06 00 B7
    SafeNet IKey4000

3B 8F 80 01 52 46 49 44 49 4F 74 20 4A 43 4F 50 20 33 36 76
    RFIDIOt JCOP 36K Blank (http://rfidiot.org)

3B 8F 80 01 80 4F 0C A0 00 00 03 06 03 00 01 00 00 00 00 6A
    Philips MIFARE Standard (1 Kbytes EEPROM)
    http://www.nxp.com/products/identification/mifare/classic/
    RFID - ISO 14443 Type A - Transport for London Oyster
    ACOS5/1k Mirfare

3B 8F 80 01 80 4F 0C A0 00 00 03 06 03 00 02 00 00 00 00 69
    Mifare card with 4k EEPROM
```

nfc-tools: mfoc

Usage: mfoc [-h] [-k key]... [-P probnum] [-T tolerance] [-O output]

h print this help and exit
k try the specified key in addition to the default keys
P number of probes per sector, instead of default of 20
T nonce tolerance half-range, instead of default of 20
(i.e., 40 for the total range, in both directions)
O file in which the card contents will be written (RE

```
mfoc -O my_01.mfd
```

```
//make a copy

cp my_01.mfd my_01_old.mfd
cp my_01.mfd my_01_new.mfd

//and take a look at them ...
```

```
hexer my_01_new.mfd

//or your favorite hex editor – shows something like -

 00000000:  7a 59 30 1c 0f 08 04 00  62 63 64 65 66 67 68 69  zY0.....bcdefghi
 00000010:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
 00000020:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
 00000030:  00 00 00 00 00 00 ff 07  80 69 ff ff ff ff ff ff  .........i......
 00000040:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
 00000050:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
 00000060:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
 00000070:  00 00 00 00 00 00 ff 07  80 69 ff ff ff ff ff ff  .........i......
 00000080:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
 00000090:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
 000000a0:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
 000000b0:  00 00 00 00 00 00 ff 07  80 69 ff ff ff ff ff ff  .........i......
```

Excurs: mfcuk

If you do not know your keys,
they are not default
and mfoc can not retrieve them,

mfcuk **MiFare Classic Universal toolKit**

can be used to brute force retrieve keys without prior knowledge.

Edit dump contents

In hex editor or in Java program, or wherever you like -

note that you have to maintain the proper structure (blocks, sectors, ..) and not all blocks will be writable.

then use nfc-tools again to write the new dump to the card:

```
nfc-mfclassic w a my_01_new.mfd my_01_old.mfd
```

Towards a working NFC app

Once you understand libnfc, use any language or environment of choice to interface it.

Alternatively, study the ACS API and sample codes to read/write your tags.

Note that you will have to write Hex ...

Sketch of a *pitcoin* card

An example from a coffee card
http://esec-pentest.sogeti.com/rfid/playing-nfc-fun-and-coffee

```
000000f0:  00 00 00 00 00 00 ff 07  80 69 ff ff ff ff ff ff  .........i......
00000100:  26 2d 34 00 00 00 00 00  00 00 00 00 00 00 00 00  &-4.............
00000110:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

```
000000f0:  00 00 00 00 00 00 ff 07  80 69 ff ff ff ff ff ff  .........i......
00000100:  26 2d 34 00 00 00 00 00  00 00 00 00 00 00 00 00  &-4.............
00000110:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

Here, in block 00000100, we have 3 numbers:

26 (in hex) = 2 * 16**1 + 6*16**0 = 38
2d ==> 45
34 ==> 52

Note that the difference is always 7 ... the price of one coffee :) - so this is an example of keeping credit and history, in a very simple way

Sketch of a *pitcoin* card

From here on, it s up to you -
you can

- encrypt
- rotate
- confuse
- keep history
- personalize
- etc etc

Make up your own scheme!