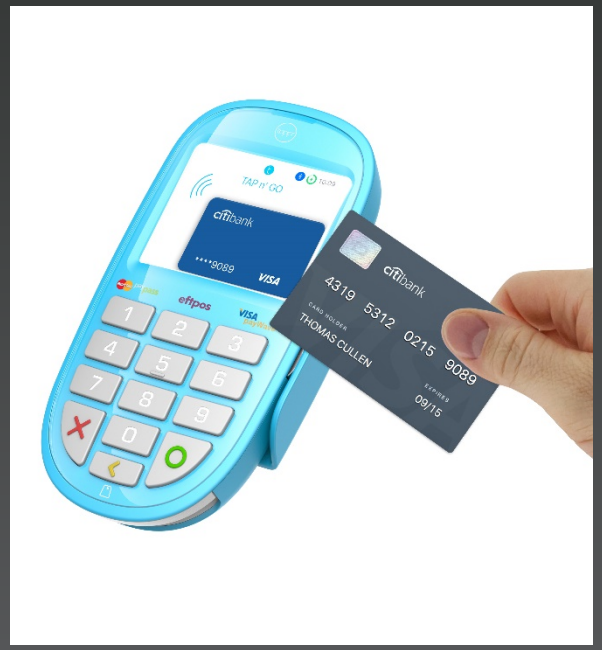
PEN TEST PARTNERS

Tag-side attacks against NFC

What is NFC?

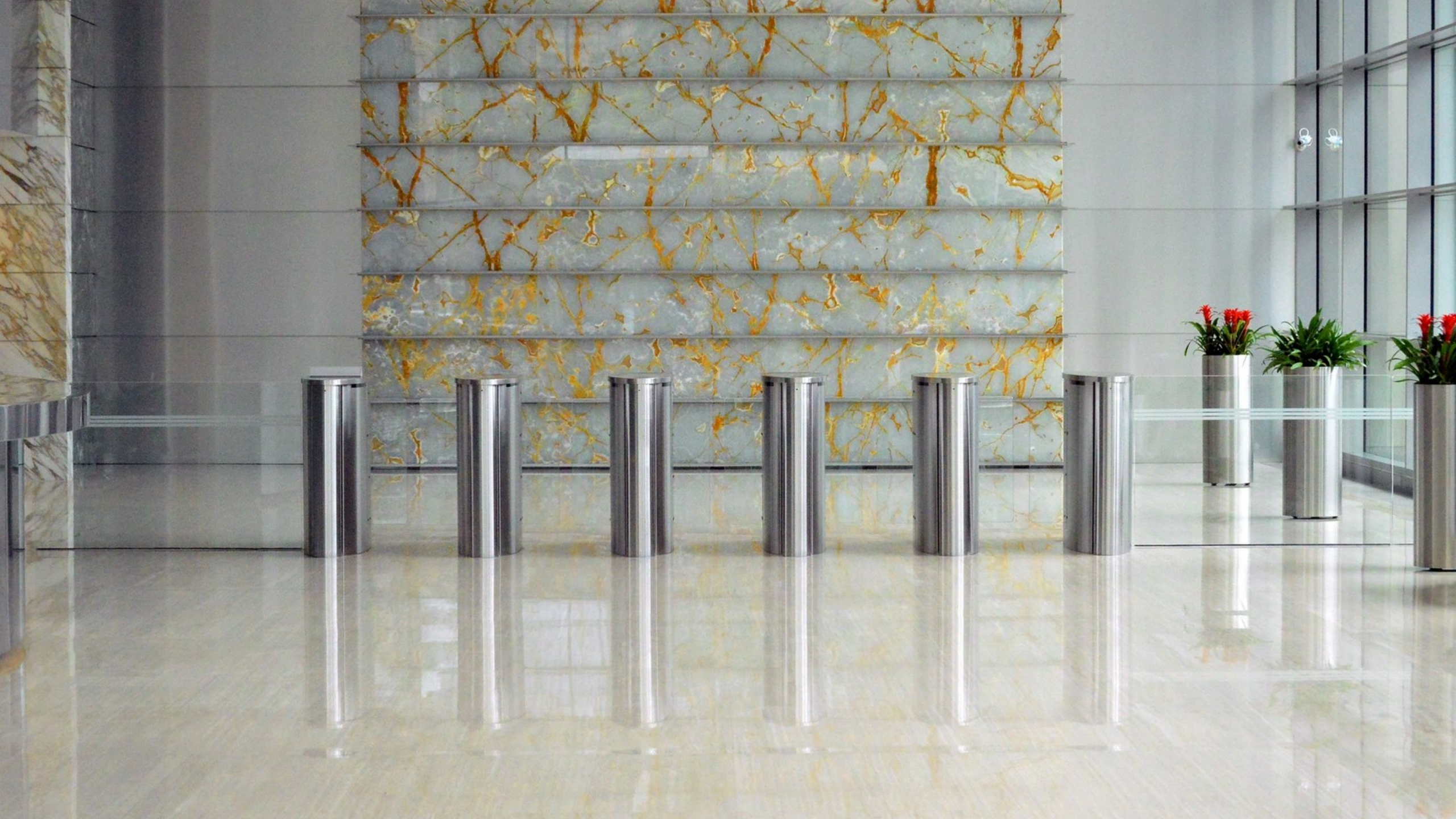
Contactless communication between two devices in close proximity

Many uses, primarily door controls and payment systems









ISO-14443

Focuses on 13.56MHz NFC communication

Defines the characteristics of the communication performed between tags and readers

Two tag types based on it, A and B

ISO-14443A is the most commonly used of the standards

Low-level communication – ISO14443A

Tags are powered by electromagnetic induction

Communication is sent by the reader by disabling the field at specific intervals

The reader uses the Modified Miller coding scheme for transmitting data to the tag

Responses are sent by the tag by modifying the power being drawn from the reader

Tags use the Manchester coding scheme to modulate the load

Each communicated byte has an additional parity bit

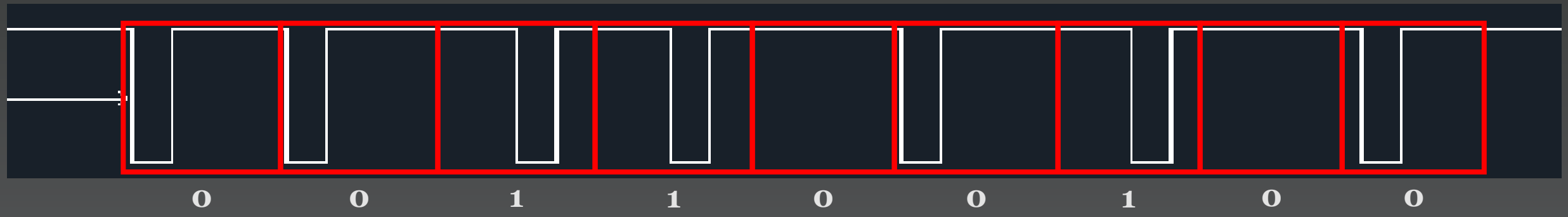
Modified Miller

Disables field a defined intervals

Minimises power loss

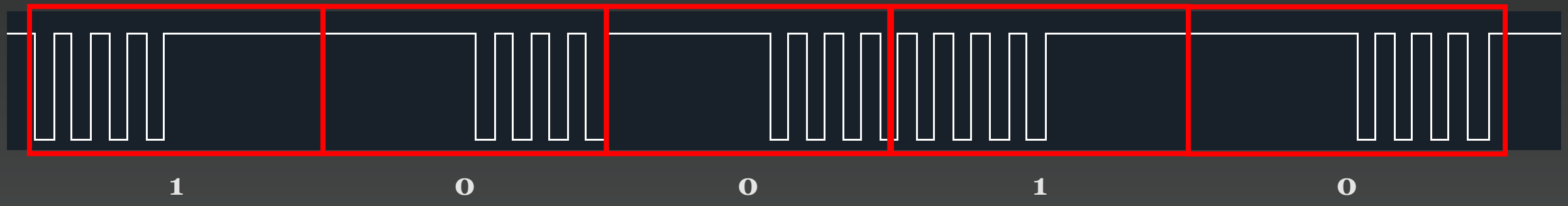
Defined as follows:

- 0 bit after 0 bit: low for the first quarter of the transmission, followed by high for the remainder of the transmission
- 0 bit after 1 bit: high for the entire transmission
- 1 bit: high for the first half of the transmission, followed by low for one quarter of the transmission, and high for the remainder of the transmission

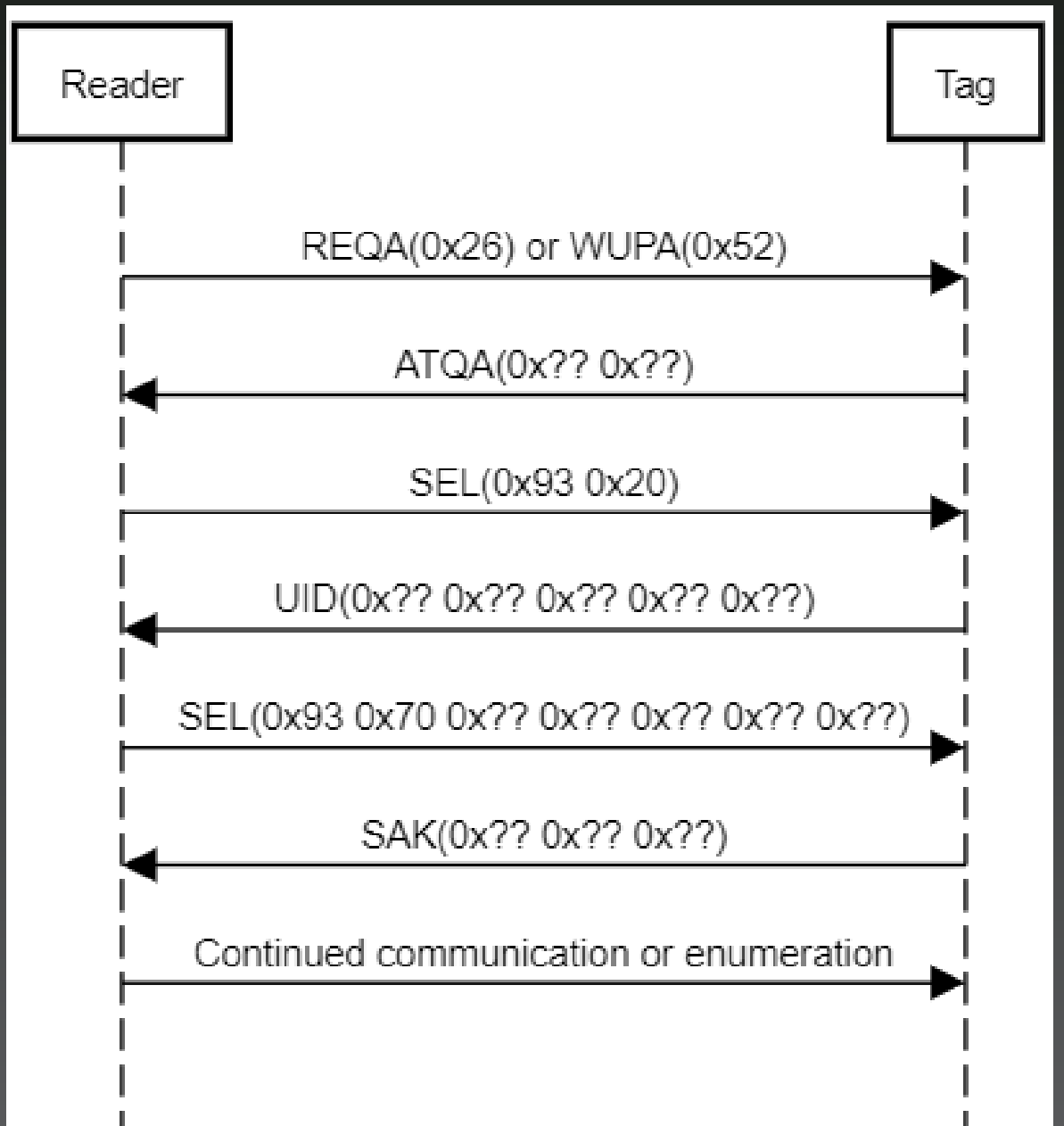


Manchester

Performed by modifying the phase of the signal
In NFC is communicated by modifying the load being drawn by the tag, using a subcarrier



Basic Enumeration



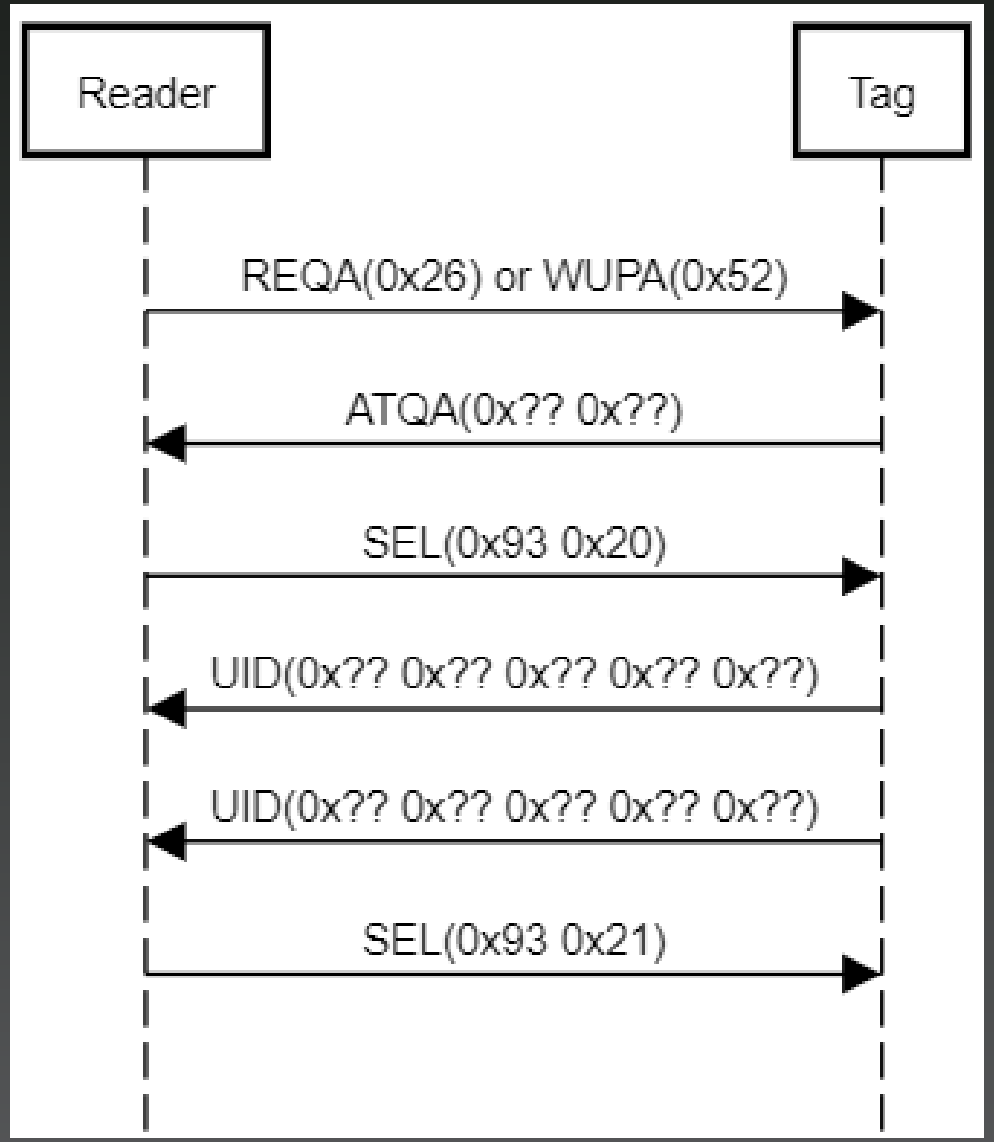
Anticollision

Performed when two tags are communicating with a reader

Involves requesting responses based on partial UIDs

Increases the number of bits requested until a single UID is identified

Once communication is complete, the next tag can be identified and communicated with



Encryption and Authentication – Mifare Ultralight

An authentication key is sent to the tag

If the key is accurate, the reader has authenticated with the tag and communication can perform

Failed attempts are logged, and in some cases can lock the chip

Can work with no authentication

Support for a wider number of authentication methods in newer versions

Encryption and Authentication – Mifare Classic

Utilises Crypto-1 algorithm

Reader requests authentication for a sector of the tag (0x60/0x61)

Tag responds with a unique four byte nonce

Reader responds with a random value, followed by an encrypted number generated from the original nonce

Tag responds with an encrypted number generated from the nonce

All further communication is encrypted as authentication has been performed

Each sector of the tag can be authenticated using its own unique keys

Encryption and Authentication – Mifare DESFire

Based on different application IDs

Authentication based on DES, 3-DES or AES depending on version and configuration

Multiple keys can be used for authentication

Authenticated similarly to Mifare Classic

Not yet been broken in any meaningful manner

Creating analysis tools

Existing tools and projects

Proxmark3 - https://proxmark.com/

Chameleon Mini - https://github.com/emsec/ChameleonMini

HydraNFC - https://hydrabus.com/hydranfc-1-0-specifications/

SimpleNFC - http://www.nonan.net/nkruse/simplenfc

Emutag - http://www.emutag.com/

NFC field detection

LED and a coil of wire

Useful for detecting when a field is active

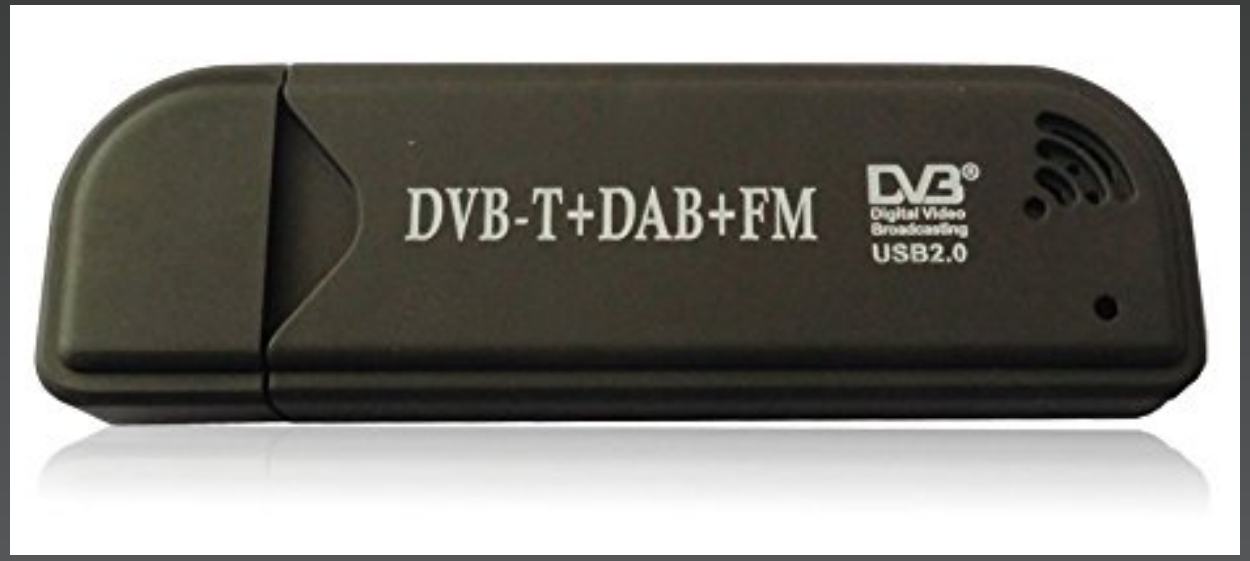


Creating a passive sniffer – RTL-SDR

Powerful SDR, able to tune between 25MHz and 1700MHz
Configurable sample rate
Libraries available for simple communication

Problems:

Cannot tune down to 13.56MHz without hardware modifications
Cannot run at a low sample rate
Bundled with weak antenna
Not built for purpose



Creating a passive sniffer – RTL-SDR

Possible to tune to harmonic frequency, providing adequate signal (27.12MHz)

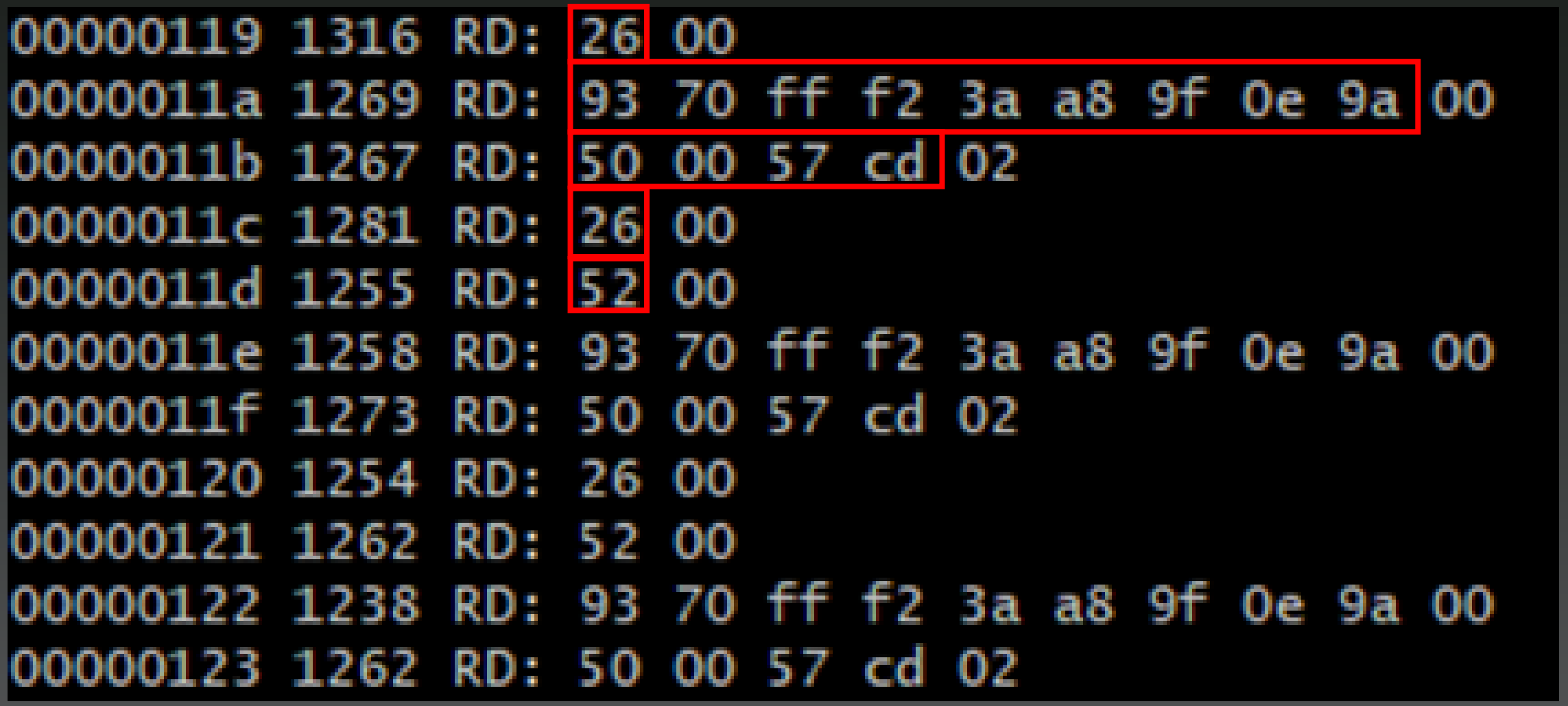
Can synchronise with the reader by setting sample rate to 1.695MHz

Antenna modifications not required, introducing a coil to the NFC field provides adequate power for analysis

Possible to detect communication from the reader to the tag

Constant signal means automatic gain control is possible

Accurate responses, providing real-time analysis of communication


00000119 1316 RD: 26 00
0000011a 1269 RD: 93 70 ff f2 3a a8 9f 0e 9a 00
0000011b 1267 RD: 50 00 57 cd 02
0000011c 1281 RD: 26 00
0000011d 1255 RD: 52 00
0000011e 1258 RD: 93 70 ff f2 3a a8 9f 0e 9a 00
0000011f 1273 RD: 50 00 57 cd 02
00000120 1254 RD: 26 00
00000121 1262 RD: 52 00
00000122 1238 RD: 93 70 ff f2 3a a8 9f 0e 9a 00
00000123 1262 RD: 50 00 57 cd 02

Building a tag

Mifare Classic was used as the initial tag type

Wanted to build it with no standard NFC chipsets, as these would limit access to raw communication

Wanted to build it with minimal components

Full implementation of Crypto-1 authentication would be required, adding complexity to the project

By fully implementing the protocol weaknesses could be identified in it

Hardware requirements

Inductive coupling would be required in order to receive signal from reader

This signal would need to demodulated by amplitude

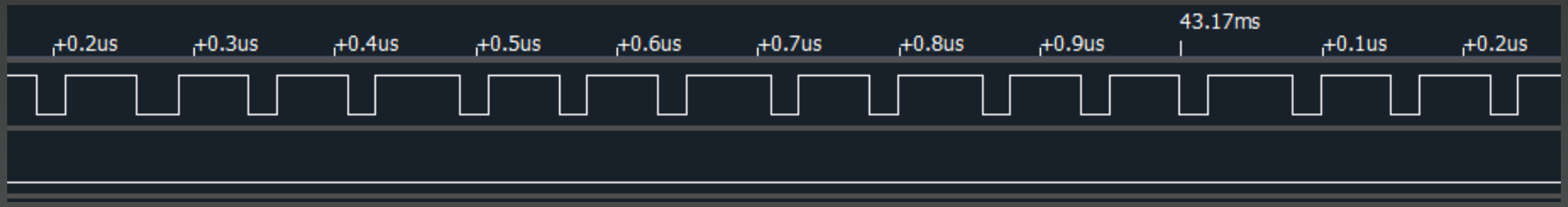
An appropriate Microcontroller which could synchronise to 13.56MHz would be required

The Microcontroller would need to be fast enough for encryption calculations

Sufficient memory would be required for storage of data

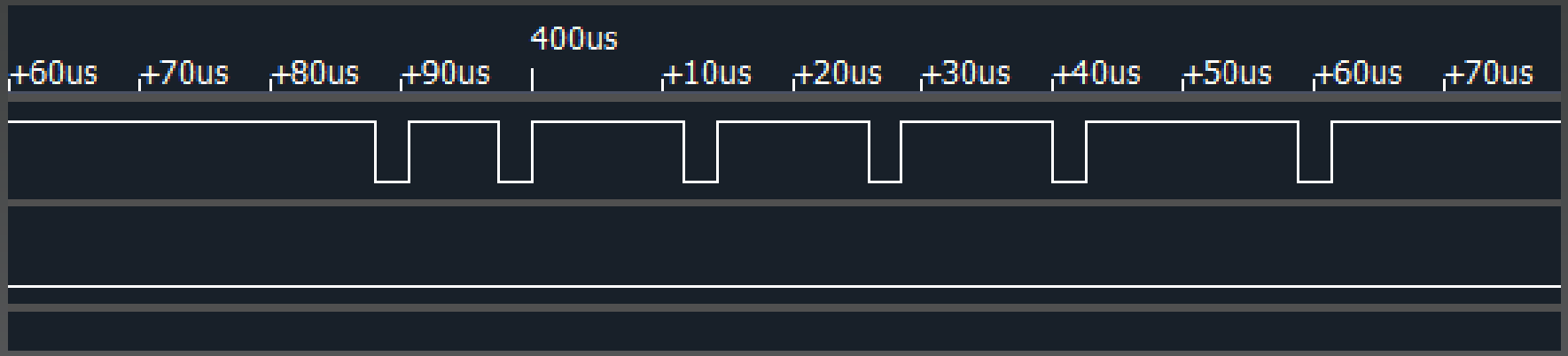
Inductive coupling - LC circuit

A circuit used to resonate with the field
Made of an inductor and a capacitor
For this purpose, a large coil of wire acts as inductor/antenna
Coil of wire was tuned to 10pF capacitor
Resonance can be checked with logic analyser, assessing whether wave frequency is 13.56MHz



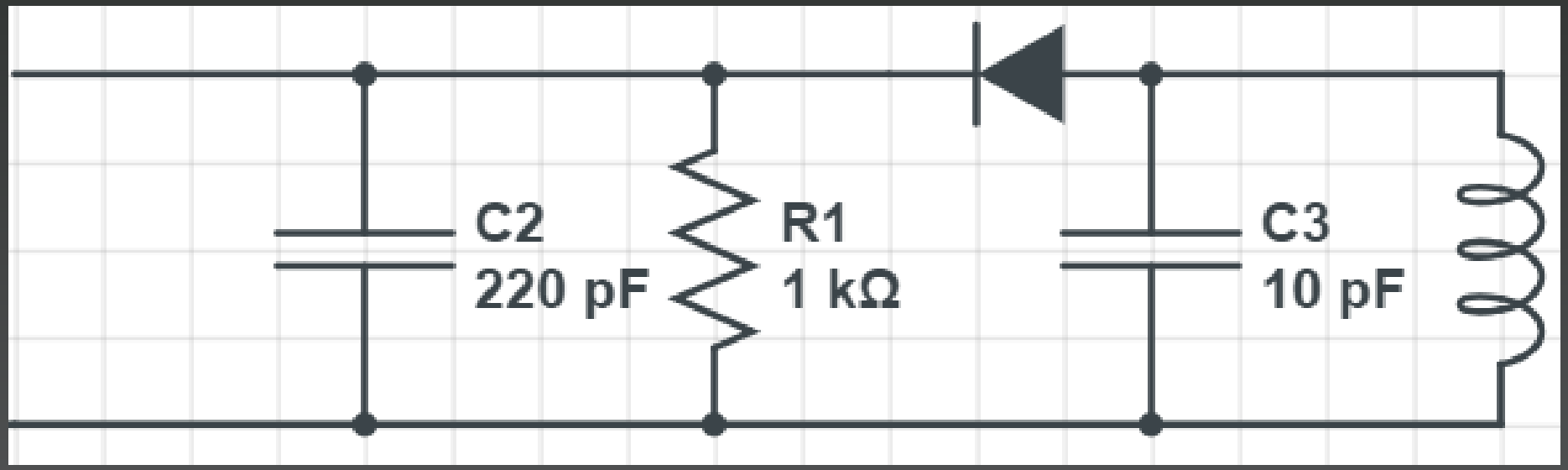
Demodulation – envelope detector

Used in circuits to demodulate signal based on amplitude
Made from a diode, resistor and capacitor
Works by rectifying the signal, and then smoothing it
Values can be guessed by trial and error
Testing concluded that a 1K resistor and a 220pF capacitor were appropriate for this circuit
Appropriate values show Modified Miller communication in logic analyser



Receiving circuit

Final circuit built with very simple layout
Accurately receives communication from a reader with sufficient voltage to trigger GPIO
No need for additional smoothing or regulation



Microcontroller selection

ATTiny84 was selected for the initial device
Utilises 8KB of program space, and 512B of RAM
Can be programmed using a standard Arduino, or dedicated programmer
Able to use external crystals to run at a specific frequency
DIP package makes it easy to build into prototypes
Receiving circuit could be attached to GPIO pins
Good support for interrupts and timers
Bad support for debugging
8-bit architecture may cause problems with fast calculation

Implementation

A 13.56MHz crystal and matching capacitors were connected to the ATTiny

Receiving circuit was connect to an input pin and output pin

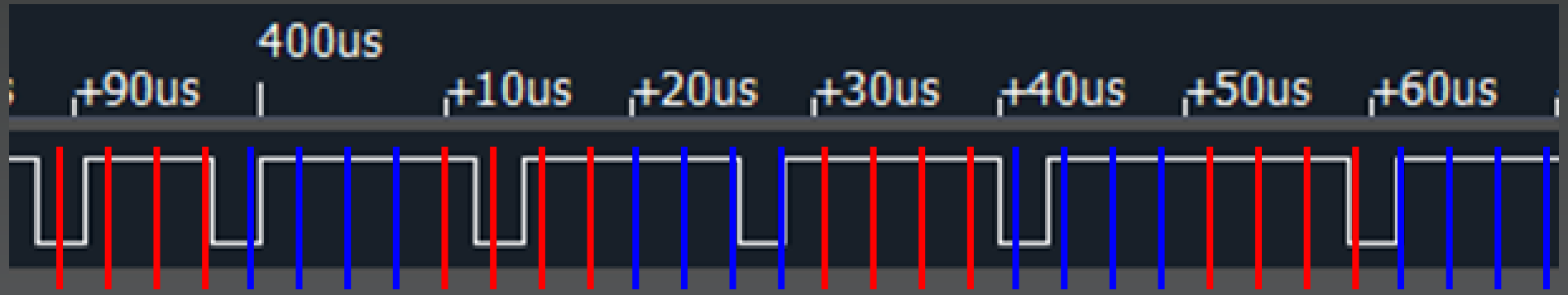
Due to UART not being feasible with non-standard clock, debug strings were communicated via software-based SPI

An LED was attached to confirm when the device was active

The responses and state machine used by Mifare Classic were implemented, allowing the device to behave as a tag

Timing issues

Using a 13.56MHz crystal, ATTiny was synchronised with reader
At predefined intervals at 847.5KHz, value of GPIO was read
Attempting to match signal against exact timings provided inconsistent results, commands were only read accurately 50% of the time
This was found to be due to clock drift on the MCU
Instead, interrupts were configure to run whenever the signal went low, and times between interrupts were assessed
This yielded a 99% accuracy



Implementing Crypto-1

Crapto-1 library and Crypto-1 papers used as a reference for implementation

Found to be based on 48-bit keys utilised as two 24-bit keys

8-bit architecture of MCU meant that all multi-byte calculations would take much longer

Additionally, AVR machine code only allows for one bit-shift at a time, making it unsuitable for cryptography

Slow responses to authentication requests would cause compatibility issues

The filter function used for all calculations was identified to be the slowest function in use

```
int filter(uint32_t const x)
{
        uint32_t f;
        f  = 0xf22c0 >> (x       & 0xf) & 16;
        f |= 0x6c9c0 >> (x >>  4 & 0xf) &  8;
        f |= 0x3c8b0 >> (x >>  8 & 0xf) &  4;
        f |= 0x1e458 >> (x >> 12 & 0xf) &  2;
        f |= 0x0d938 >> (x >> 16 & 0xf) &  1;
        return BIT(0xEC57E80A, f);
}
```

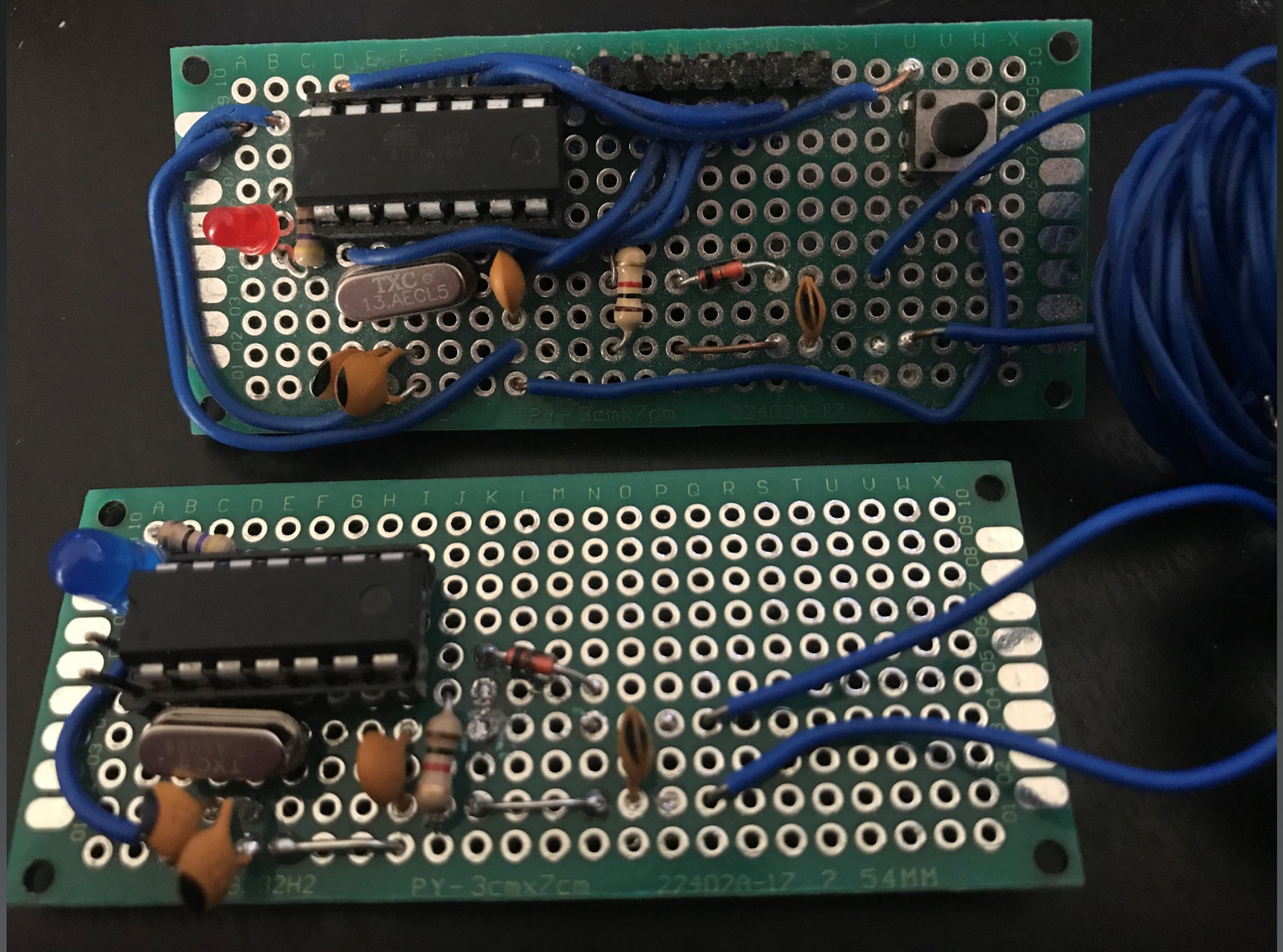
Implementing Crypto-1

All Crypto-1 code was converted from C to AVR assembly
SimAVR was used to compare optimised assembly version against C version in environment with more effective debugging
All calculations which would be treated as 32-bit were performed as 24-bit operations
All bit-shifts were converted into more efficient operations:

- 16-bit shifts - move the two upper bytes to the lower two bytes
- 8-bit shifts - move the second lowest byte to the lowest byte
- 4-bit shifts – use the AVR SWAP operation to swap the upper and lower nibbles
- 2-bit shifts – two traditional shift operations
- 1-bit shifts – one traditional shift operation

These optimisations increased the speed of the calculations by ~10 times

```
        breq endShiftSixteenBit
        mov r16, r26 ; this will shift 16 bits in two operations rather than the ton it usually takes
        mov r17, r27
endShiftSixteenBit:
        ldi ZH, hi8(secondLookupTable) ; set up first space for lookup
        ldi ZL, lo8(secondLookupTable)
        add ZL, r21
        adc ZH, r1
        lpm r21, Z
        cpi r21, 0
        breq endShiftEightBit
        mov r17, r16
endShiftEightBit:
        ldi ZH, hi8(firstLookupTable) ; set up first space for lookup
        ldi ZL, lo8(firstLookupTable)
        add ZL, r23
        adc ZH, r1
        lpm r23, Z
        cpi r23, 0
        breq endShiftFourBit
        swap r17
        andi r17, 0x0f ; may not be necessary
endShiftFourBit:
        ldi ZH, hi8(firstLookupTable)
        ldi ZL, lo8(firstLookupTable)
        add ZL, r20
        adc ZH, r1
        lpm r20, Z
        cpi r20, 0 ; if its zero, skip
        breq endShiftTwoBit
        lsr r17
        lsr r17
endShiftTwoBit:
        ldi ZH, hi8(secondLookupTable)
        ldi ZL, lo8(secondLookupTable)
        add ZL, r24
        adc ZH, r1
        lpm r24, Z
        cpi r24, 0
        breq endShiftOneBit
        lsr r17
endShiftOneBit:
        andi r17, 0x01
        mov r24, r17
        pop r16
        pop r17
        pop r18
        ret ; make sure r24 contains 8 bit response value - actually a 1 bit response
```


TXC
13.AECL5
PY-3cmx7cm
224020-17
PY-3cmx7cm
224020-17 2 54MM



4108516232

4100587656

4102771336

4097162120

4105371528

4108055176

4098583176

4104376456

4105179528

4104552584

Hardware limitations

Microcontroller is too slow to perform complex operations in time, 13.56MHz clock speed is not enough to perform additional functions

~400 bytes of RAM and ~7000 bytes of Flash were used for the implementation, leaving little room for further functionality

512 bytes of EEPROM memory were not enough to store an entire tag

Debugging complex functionality on AVR Microcontrollers is a difficult process

Limited number of pins means limited number of additional peripherals could be added

Some readers were still not compatible, as they require fast response times

Building a better device

A more powerful Microcontroller was selected – STM32L496ZG

1MB of Flash available, 320KB of RAM (640x more than the ATTiny)

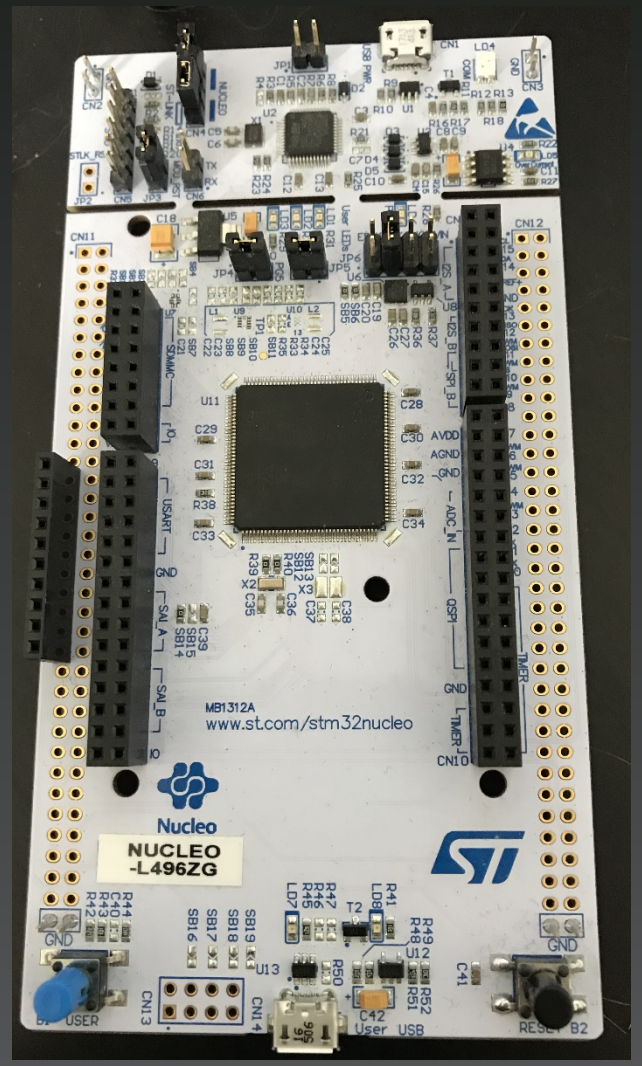
Much faster clock speed, capable of performing at 72MHz

32-bit architecture could improve encryption calculations

Large number of useful peripherals, including USB and UART

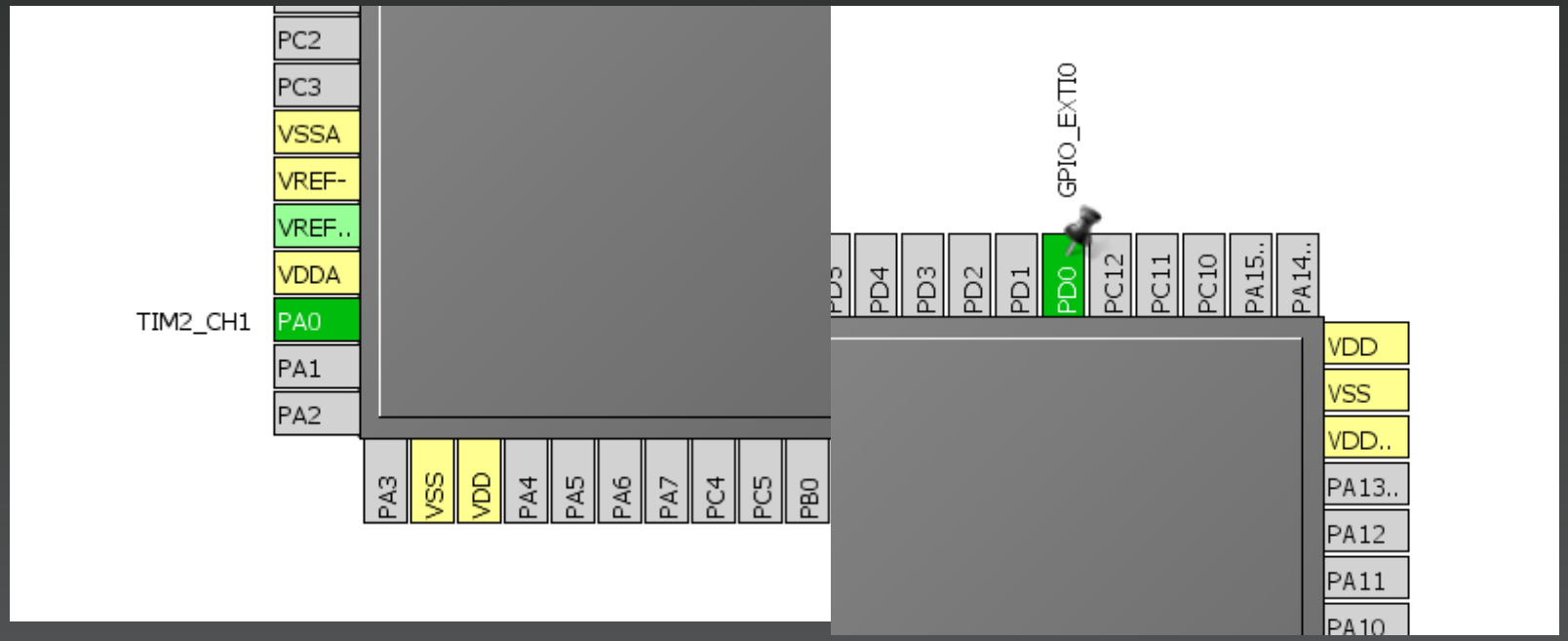
Easy to program and debug using development software

Can be built using the same circuitry and logic as previous device



Building a better device

Pin configurations and clock speeds can be set in STM32Cube
Very little modification to core codebase was required – only timers and interrupts needed to be modified
Synchronisation issues could occur due to lack of standard clock rate



Building a better device – Synchronisation

Internal clock of the chip was not able to tune to 13.56MHz

This would not be a problem for receiving data, but would be a problem for transmitting responses

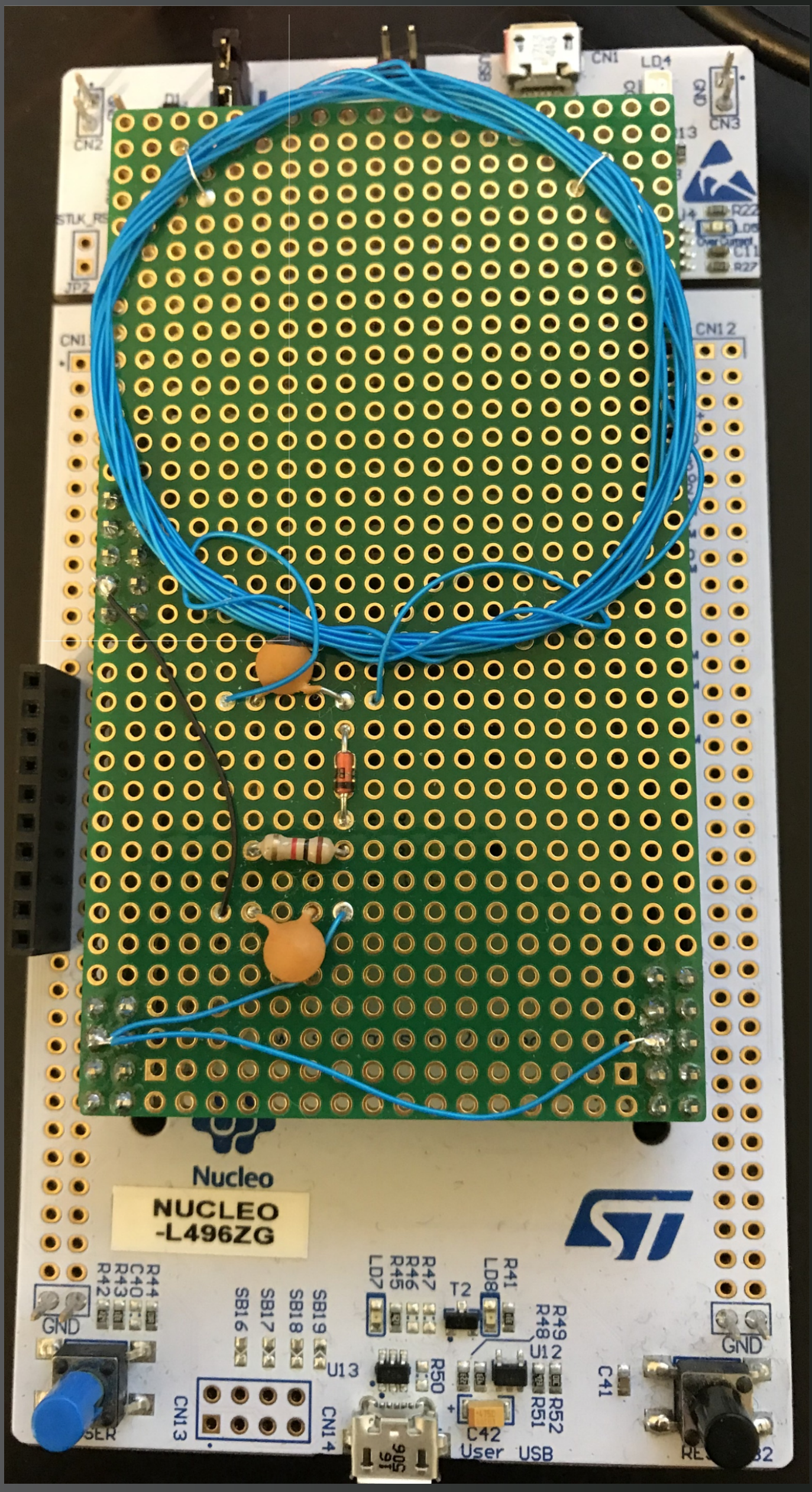
Use of an external crystal was not ideal, as it would increase complexity of the board

Without an accurate clock, the device would fall out of sync with the reader

The STM32 can run at a large number of frequencies, it could be possible to find a clock at a close enough frequency, without being perfectly in sync

Each possible frequency was assessed in order to find the most suitable candidate

```
72000000 - 0.044247787610615
67000000 - 0.05604719764011 2
61000000 - 0.023598820058993
56000000 - 0.07669616519173 4
50000000 - 0.00294985250737 77
39000000 - 0.017699115044245
33000000 - 0.061946902654867
28000000 - 0.038348082595867
22000000 - 0.041297935103245
17000000 - 0.058997050147493
11000000 - 0.020648967551622
6000000 - 0.079646017699115
```


Nucleo
NUCLEO
-L496ZG
CN1
CN2
CN3
CN12
LD4
LD7
LD8
T2
U13
U1
C42
User USB
GND
CN13
CN14
SB16
SB17
SB18
SB19
C41

Additional features – Multi-tag support

Can be implemented by having several instances of the same state machine

Handling of multiple requests can be performed by forcing anticollision, or more easily by cascading requests so that only one selection is performed at a time for each emulated tag

This can be used to exploit weaknesses related to selecting multiple tags

Not hugely useful, as readers rarely implement this functionality

Additional features – Dynamic Crypto-1 Key Generation

Some readers generate unique keys depending on the UID of the tag

If the algorithm for these can be reverse engineered, keys can be calculated as authentication is requested

This can allow for dynamic modification of UID values without adversely affecting authentication

Additional features – Implementing DESFire

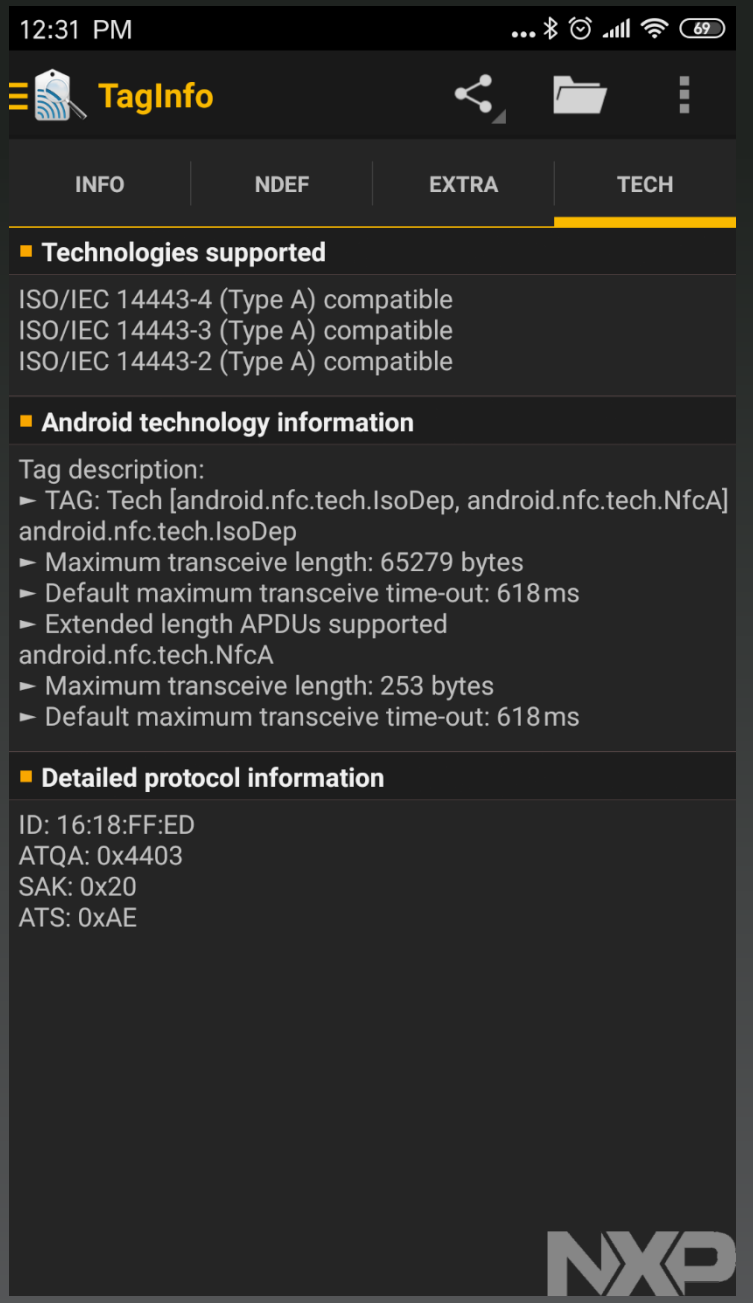
Can present as DESFire by modifying SAK and ATQA response values

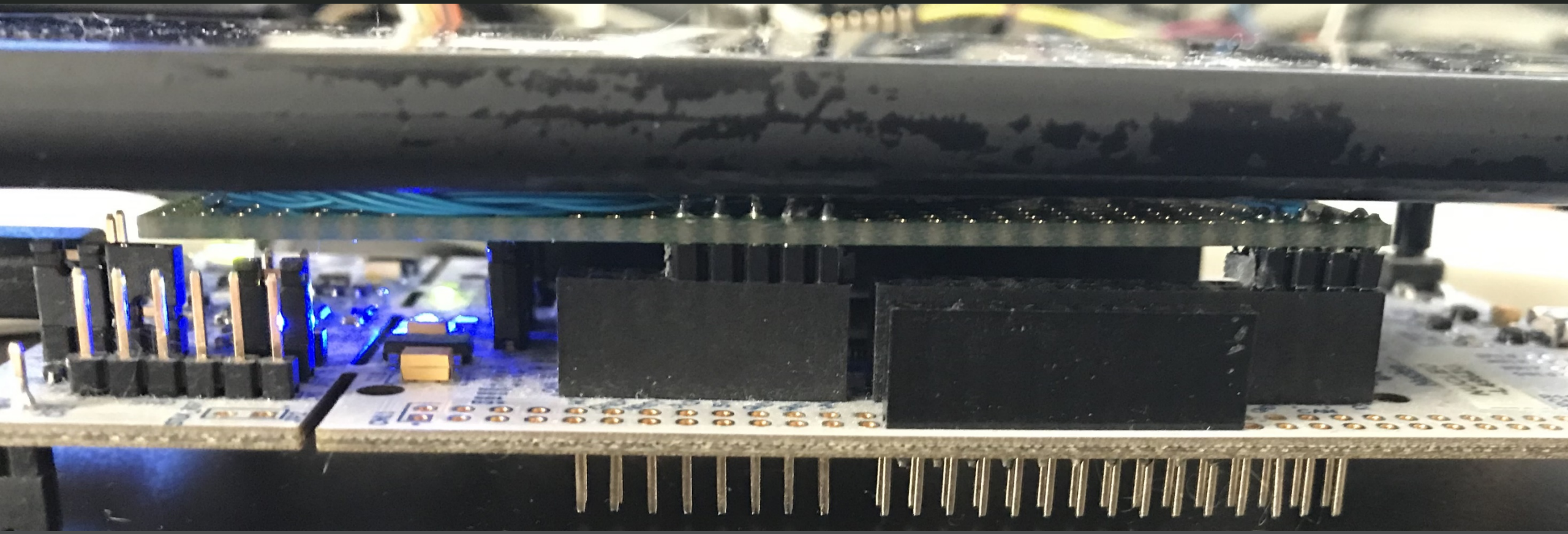
Implemented by replaying legitimate requests from readers

Protocols are robust, when a valid response can't be provided, the reader will try again

Authentication functionality is well documented

The Mifare DESFire Tool Android Application can be used to develop and test this functionality





Security Weaknesses

Crypto-1 Weaknesses

Weaknesses in this algorithm have been known for a long time, most importantly in the paper “Dismantling Mifare Classic”, which outlines the most key weaknesses.

Crypto-1 utilises 48-bit keys, split into 24-bit keys, this can be brute forced

Authentication is vulnerable to replay attacks

Nonces used to authenticate can be used to recover 32-bits of keystream from authentication

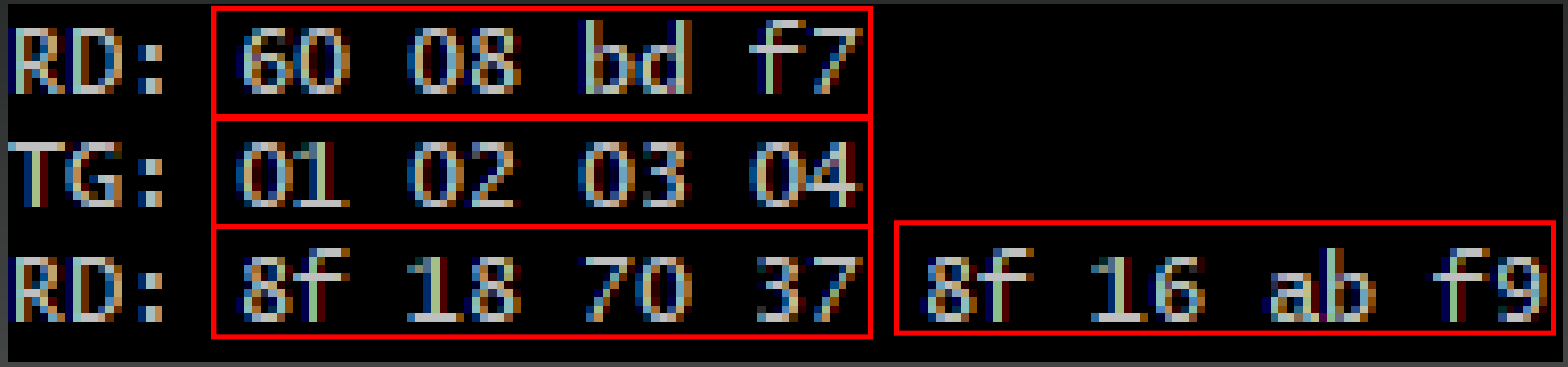
Rollbacks from the authentication can be performed to fully recover the initial key

Attacking Crypto-1 from the tag

Key recovery for a sector can be achieved from two authentication requests when emulating a tag:

- Authentication is allowed to progress until the reader sends a response to the tag's initial nonce
- This response contains a random value followed by an encrypted value generated by the tag's nonce, the generated value can be calculated and XORed with this, providing 32-bits of keystream
- The two 24-bit keys generated from the 48-bit initial key are used to calculate each alternating bit of the keystream, reducing these values from ~16 million possible 24-bit keys to ~200,000
- Matching key pairs can be checked in order to find combinations which generate the 32-bit keystream
- These keys can be rolled back through the random value and the initial nonce in order to recover the initial key

Attacking Crypto-1 from the tag



Attacking Crypto-1 from the tag

This approach can be used to perform offline cracking on Mifare Classic keys

Keys can be recovered in under ten minutes

This is more efficient than reader-based attacks on tags, but is impractical in a real-world setting

This functionality is available on the Proxmark and Chameleon Mini, and is part of the core functionality of Crapto-1, but is not widely used to due to practicality in most contexts

Attacking Crypto-1 from the tag - Demonstration

A Mifare Classic reader with no known research was selected – the NFC reader used by a Japanese Video Game

Reader was identified to use USB for communication, which was reverse-engineered using the USBProxy tool and a Beaglebone Black, the protocol was found to be simplistic and allowed for access to tag UIDs and block data

The reader was found to not be compatible with the Proxmark or Chameleon Mini due to the speed with which it required responses from tags

A tool was written which identified tags on the reader and tried to read data from it


SOLDERING STATION
MB1312 A-01
217310470
T1 T2 T3 T4 QA

Attacking Crypto-1 from the tag – Demonstration

UIDs, sector numbers and authentication values were generated from the STM32 device
A custom tool was written which processed these in order to recover keys
Keys were verified against the reader after being generated, showing that they could be recovered
A large number of different keys were generated by emulating a large number of unique UIDs, this allowed for identification of differences between keys, and could help with reverse engineering key generation algorithms

```
Figure placement
UID Update, reading block: 00 - c6 7e f4 a7
Data: c6 7e f4 a7 20 14 a7
Figure placement
Figure placement
UID Update, reading block: 01 - c6 7e f4 a7
Data: c6 7e f4 a7 20 14 a7
Figure placement
Data: c6 7e f4 a7 20 14 a7
Data: c6 7e f4 a7 20 14 a7
Data: c6 7e f4 a7 20 14 a7
```

```
02 a7 20 14 a7 01 02 03 04 6c 30 cf 0c 1c 96 be bb
02 a7 20 14 a7 01 02 03 04 a3 a8 ba 52 79 f9 6e 2d
02 a7 20 14 a7 01 02 03 04 d3 68 be 98 ae ba 26 09
02 a7 20 14 a7 01 02 03 04 1a 97 72 34 0f 73 95 94
02 a7 20 14 a7 01 02 03 04 dd 57 d5 4e 11 0f 48 4d
02 a7 20 14 a7 01 02 03 04 af b3 8b b4 c7 d5 8f 82
02 a7 20 14 a7 01 02 03 04 3f 6c dc 3d 32 33 68 86
02 a7 20 14 a7 01 02 03 04 73 b9 c8 6d 80 7f ad e9
02 a7 20 14 a7 01 02 03 04 7a 46 a9 f6 28 c5 9f e4
02 a7 20 14 a7 01 02 03 04 44 a0 a6 de 97 31 0a fa
```

```
Sector: 02 Uid: a7 30 10 a7 nonce: 01 02 03 04 resp:  ed d9 5d 51 7f bb 67 62
Successor: 04030201->56edf820->adcd2b3c
Sector: 02 Uid: a7 30 10 a7 nonce: 01 02 03 04 resp:  ed 2e da f1 23 55 63 13
Successor: 04030201->56edf820->adcd2b3c
Sector: 02 Uid: a7 30 10 a7 nonce: 01 02 03 04 resp:  83 c8 9e cc f5 95 29 ae
Successor: 04030201->56edf820->adcd2b3c


Total potentials 0 - 23538(0) 31492 - 0
Got working matched on subsequent at: c5afe6 ca6cd8
Test key: 29 6f af 36 99 0f

real    9m58.366s
user    9m58.283s
sys     0m0.083s
```

Crypto-1 - Improvements

An increased key size would significantly increase the complexity of the attack

Usage of a single, large key would prevent brute forcing of key stream values

An improved PRNG on the tag would limit replay attacks

Removing known plaintext from authentication would remove opportunity for exploitation

Raw protocol weaknesses

Most NFC chipsets don’t support control of the initial enumeration procedures, leaving it only accessible for testing by dedicated devices

Initial enumeration is performed by all tag types, meaning that weaknesses can be tested on a large number of readers

No limits on response sizes leave readers potentially open to memory corruption weaknesses

The greatest weaknesses lie in the anticollision procedures

Combine the known bytes and the received bytes to form the complete UID.

1. Reset bit B7 (B7 = 0) of the ISO control register to enable reception with CRC.
2. Send a SELECT command with the appropriate cascade level, NVB = 0x70 and complete UID.
3. Wait for End of TX interrupt.
4. Wait for End of RX interrupt.

   Read the data (received from the tag) in the FIFO. Examine the SAK field in the tag response. If the cascade bit is set, UID is not complete. Therefore, increase the cascade level and initiate a new anti collision sequence.

Note: Due to the recursive nature of the anti-collision algorithm, there is a risk of stack overflow. It is highly recommended that the user implement stack overflow check in the firmware.

The anti collision loop of the ISO14443A protocol is described in more detail in the flowchart below.

Exploiting anticollision

By constantly responding to requests with corrupted communications, the reader will ask for an increasing number of bits of the UID, eventually overflowing and requesting a size too large for the buffer

This can often cause crashes

This feature is rarely implemented, most readers only support one card at a time

This weakness is known, but is still found in some readers

```
00000096 3532 RD: 93 07 ff ff ff ff ff ff ff ff ff ff ff 7f 00
00000097 3536 RD: 93 00 ff ff ff ff ff ff ff ff ff ff ff ff 00
00000098 3530 RD: 93 01 ff ff ff ff ff ff ff ff ff ff ff ff 01
00000099 3437 RD: 93 03 ff ff ff ff ff ff ff ff ff ff ff ff 03
0000009a 3312 RD: 93 03 ff ff ff ff ff ff ff ff ff ff ff ff 07
0000009b 3434 RD: 93 07 ff ff ff ff ff ff ff ff ff ff ff ff 0f
0000009c 3434 RD: 93 05 ff ff ff ff ff ff ff ff ff ff ff ff 1f
0000009d 3366 RD: 93 06 ff ff ff ff ff ff ff ff ff ff ff ff 3f 00
0000009e 3379 RD: 93 07 ff ff ff ff ff ff ff ff ff ff ff ff 7f 00
0000009f 3618 RD: 93 00 ff ff ff ff ff ff ff ff ff ff ff ff ff 00
000000a0 3889 RD: 93 01 ff ff ff ff ff ff ff ff ff ff ff ff ff 01
000000a1 4215 RD: 93 03 ff ff ff ff ff ff ff ff ff ff ff ff ff 03
```

High level protocol weaknesses

Each tag type has its own weaknesses

Entire stack lends itself to fuzzing beyond enumeration, as this data can be manipulated by most NFC chipsets

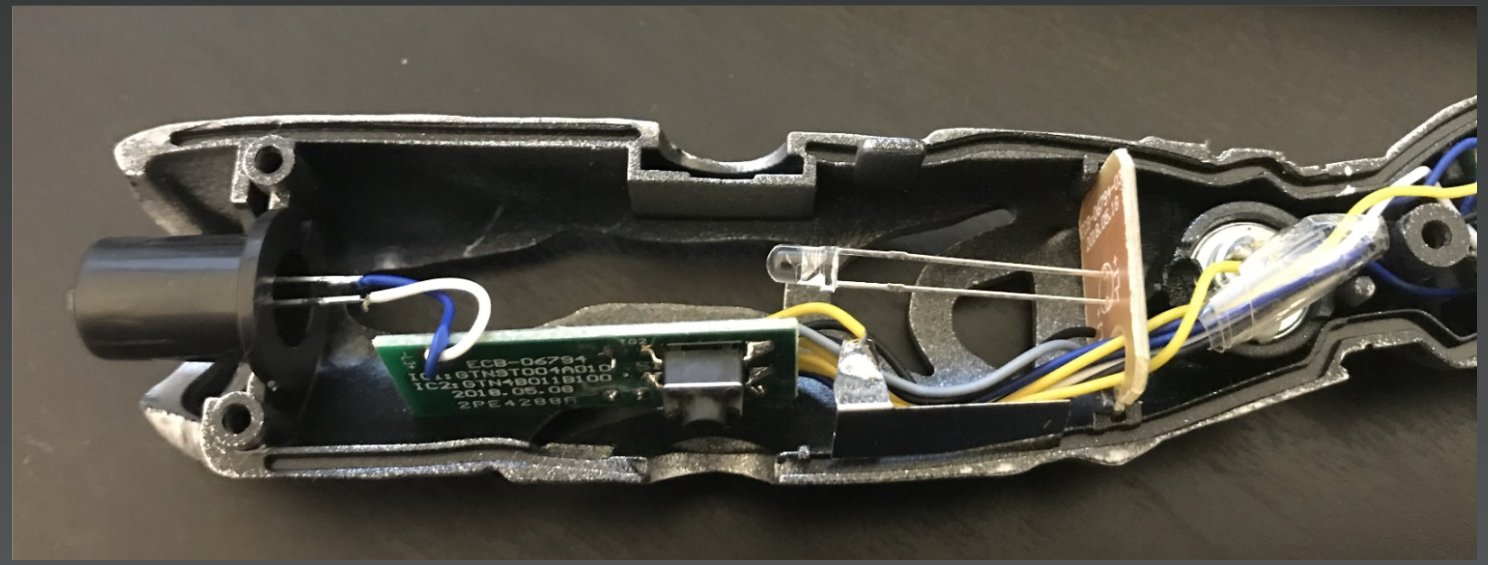
NDEF data has the greatest potential for weaknesses, as it has a large number of data types and features

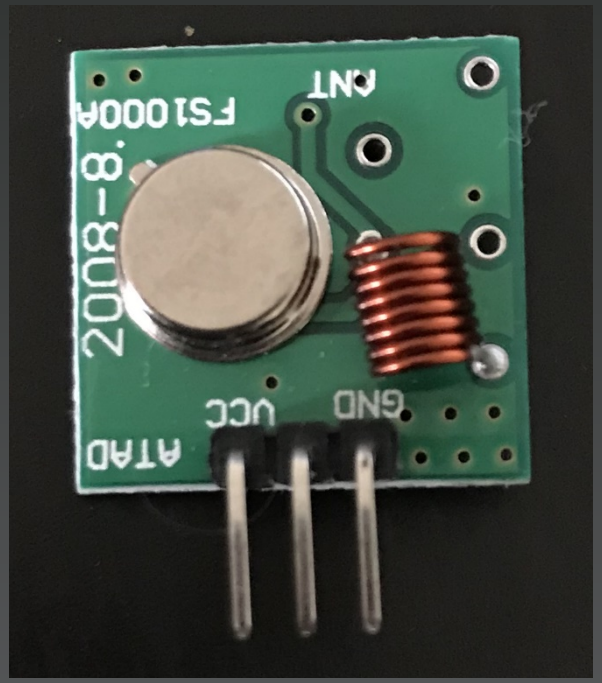
Authentication mechanisms often have known plaintext weaknesses

Capabilities of tag hardware mean that complex authentication and encryption is not possible

Compiling research







Future work

Source will be released – RTL-SDR, AVR and STM32 tools

Boards will be designed

DESFire will be fully implemented, and each element of it will be assessed

More tag types will be researched for weaknesses

Questions

PEN TEST PARTNERS

Christopher Wade
@Iskuri1
https://github.com/Iskuri
https://www.pentestpartners.com