



Uncovering Vulnerabilities in Hoermann BiSecur

An AES Encrypted Radio System

Markus Müllner, Markus Kammerstetter, Christian Kudera and Daniel Burian

Trustworks KG

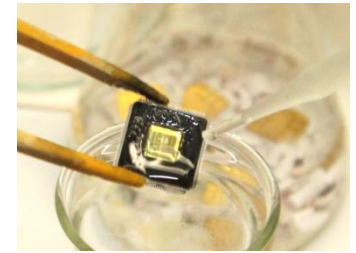
Web: <https://www.trustworks.at>

E-Mail: office <at> trustworks.at

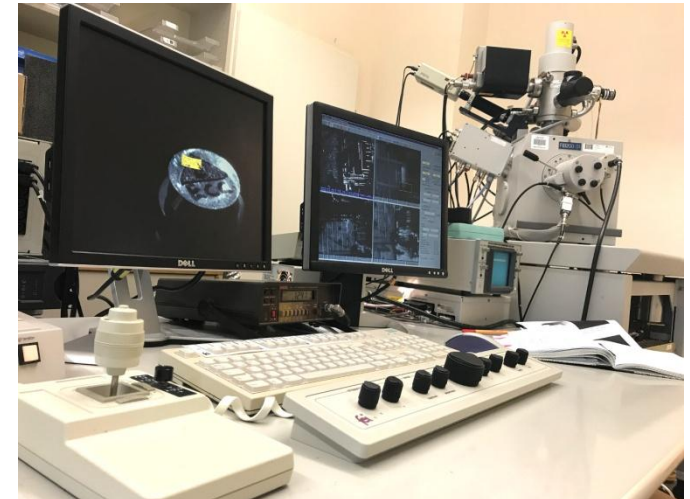
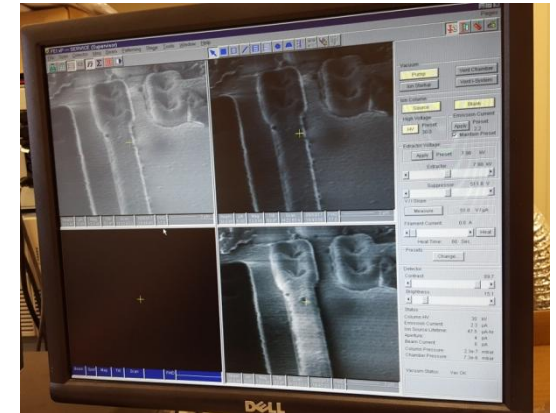
Who we are

- Security consulting, engineering and research company, founded 2012
- Security consulting services since 2005
- Core business area:
 - Security audits of Web-applications and enterprise software products down to embedded devices and microchips
 - Embedded & Security Engineering
 - High-speed cryptography
- Own Hardware Security Lab to support embedded security audits

Our Hardware Security Lab



- Dedicated lab equipment such as FIB and SEM to conduct embedded security audits down to the microchip level
- Non-invasive (SCA, FI) to fully-invasive (FIB/SEM) evaluation methods
- General idea: More advanced lab tools reduce required analysis time during security audits



Why Garage Doors ?

- Many wireless garage door systems are known to be insecure:
 - Mostly static or simple rolling code schemes
 - No encryption
 - Replay attacks and cloning possible
- Hoermann BiSecur:
 - Use of AES algorithm, established high-security system
 - Big security improvement over “classical” systems
 - Two of our security analysts already had a BiSecur system at home

Open Questions ...



We have those systems at home, are they really secure ?

- Is the system implementation secure in practice ?
 - How does the system use AES encryption ?
 - How is the key material generated and used ?
 - Is the key material individual ?
 - What messages are exchanged on the RF interface ?
 - Kerckhoff's principle: Is the system secure if all system internals except for the key material are known ?

→ We decided to conduct a security audit

Hand Transmitters

- We already had a few
- We also obtained new ones
- Result:
 - Different models
 - Different manufacturing dates
 - Our analysis will cover a broader range



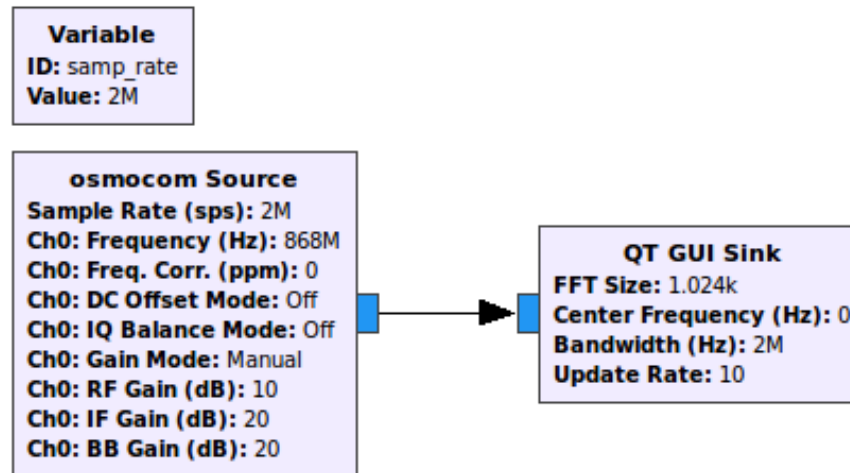
RF Signal Analysis

- Manual: Signal at 868 MHz
- Use of BladeRF SDR
- Goals:
 - Find exact frequency
 - Identify modulation scheme
 - Identify channel coding
 - Decode RF frames

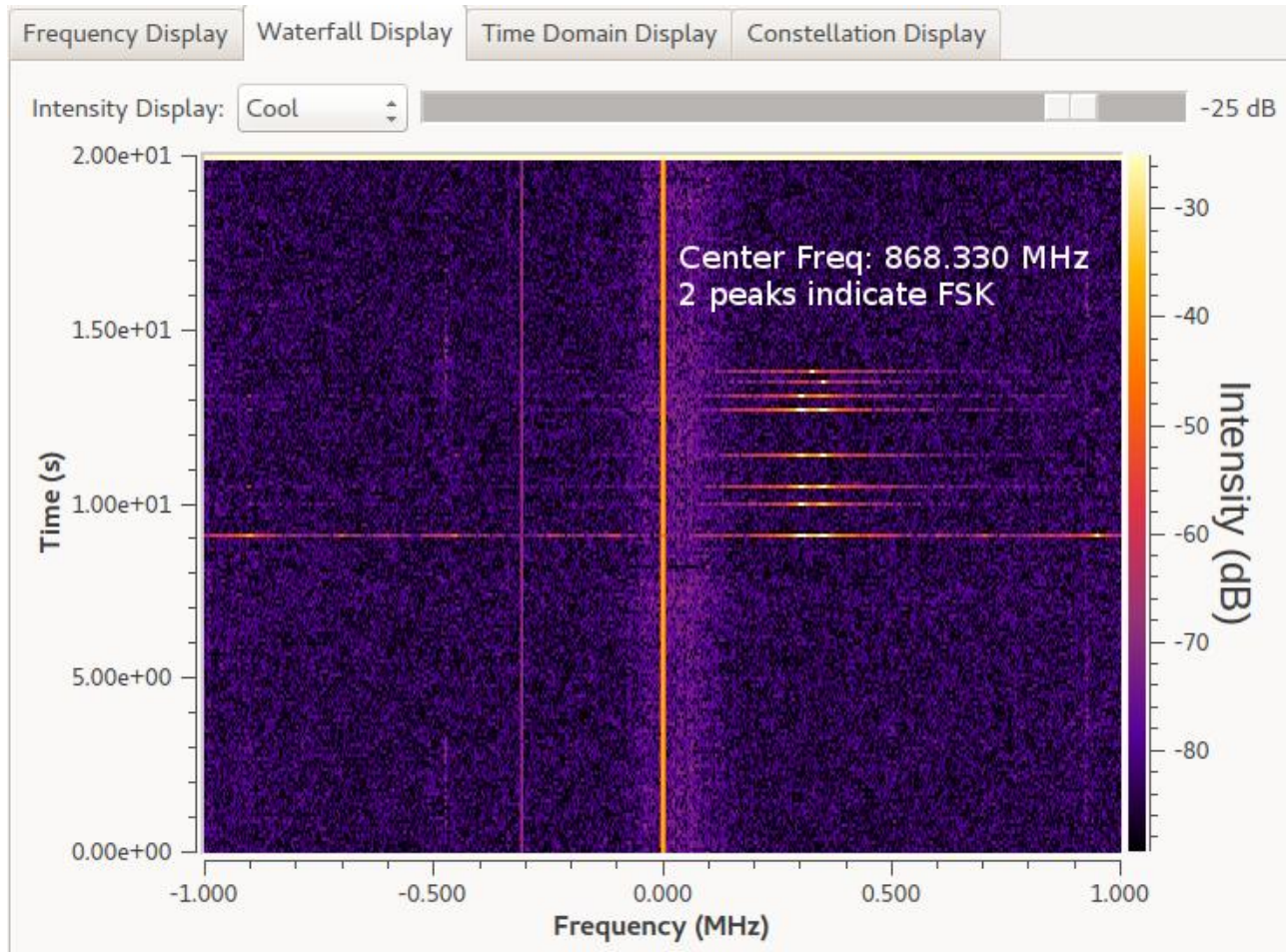


Finding the Exact Frequency

- GNU Radio SDR Suite
- Spectral analysis (Waterfall Plot)



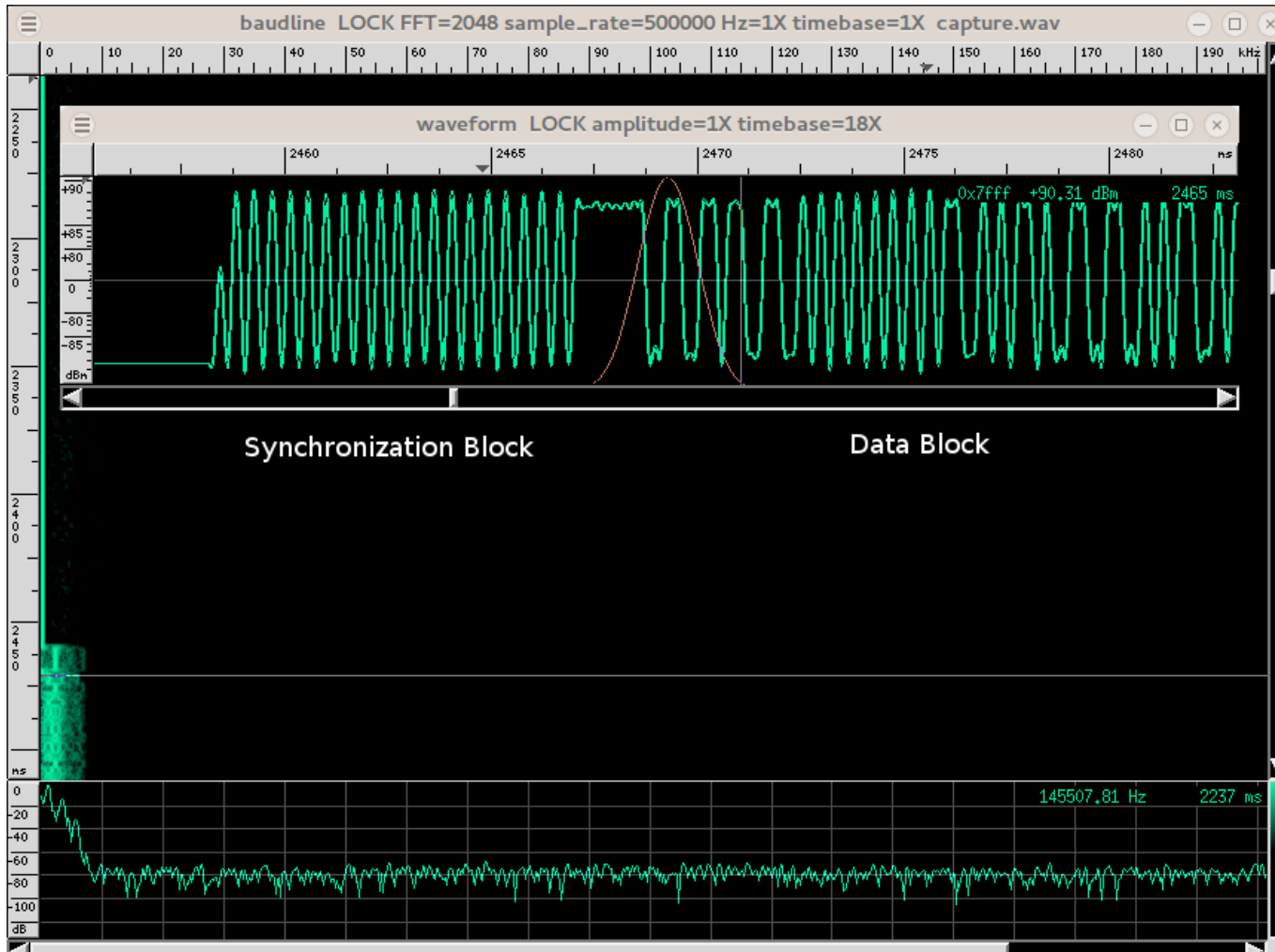
Finding the Exact Frequency



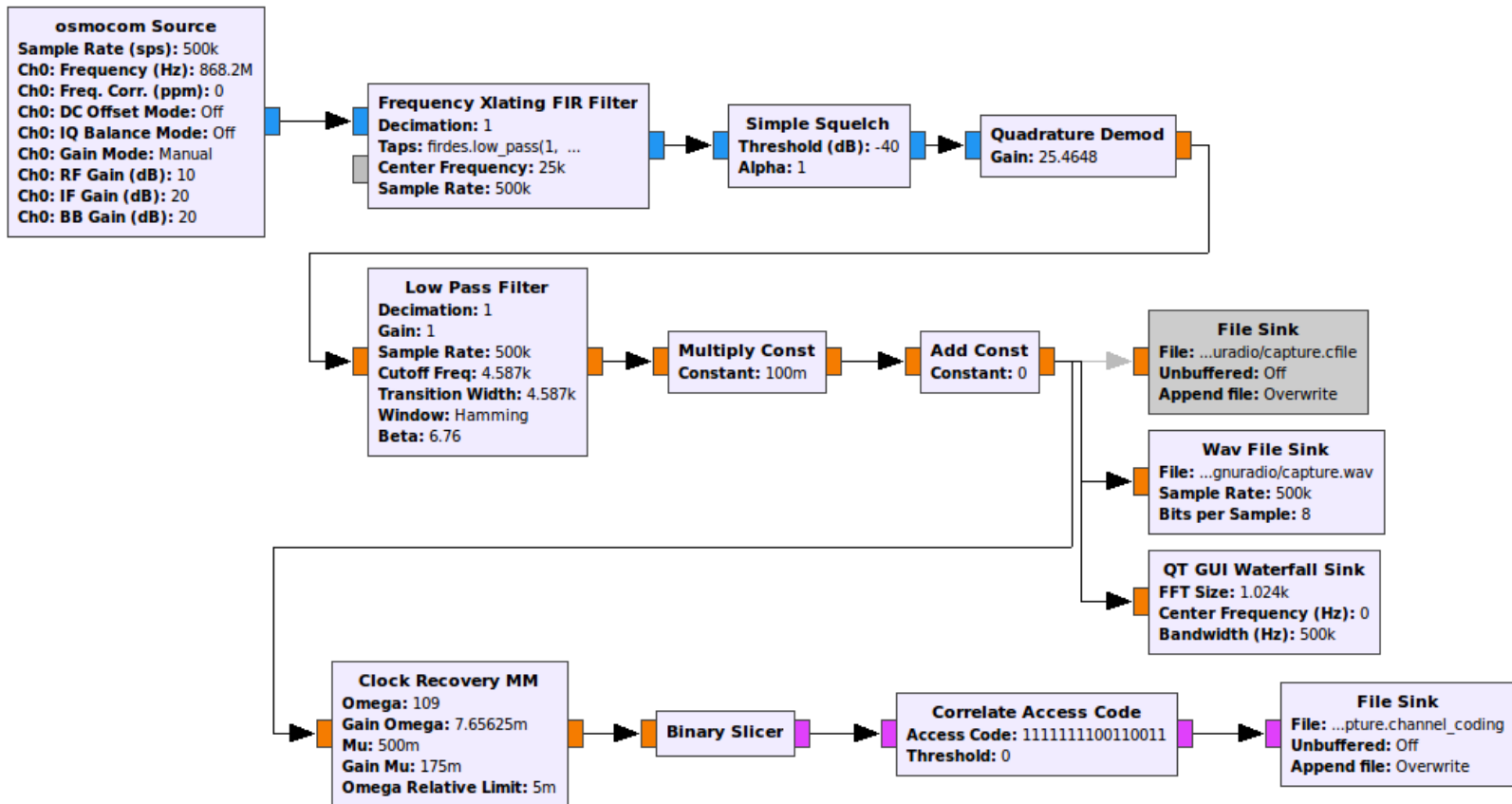
Identify Channel Coding

- Typical transmissions contain a synchronizing block (i.e. alternating high/low waveform)
- Allows receiver to synchronize its symbol rate to the symbol rate of the transmitter
- Allows us to determine the symbol rate as well

Identify Channel Coding

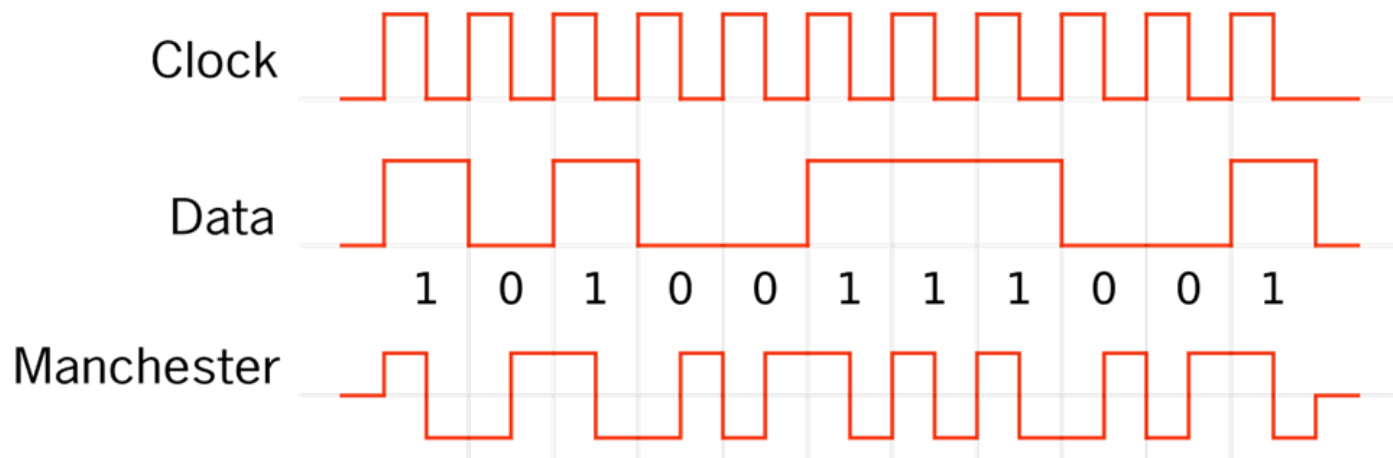


Identify Channel Coding



Channel Coding Result

- We analyzed the captured dump file
- Demodulated signal contains either '01' or '10' sequences --> Manchester encoding
- 2 symbols represent 1 bit of data



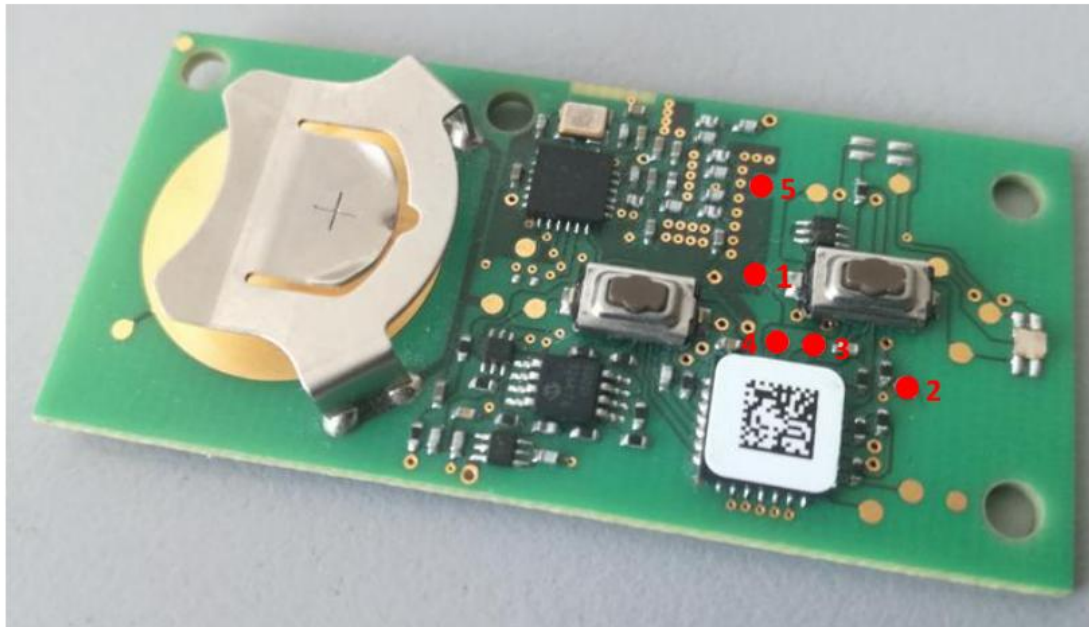
[Wikipedia, Manchester Code]

Decoding RF Frames

- Decoding of data bits from recorded frame (Python script with GNU Radio framework)
- Assumption: CRC value at end (unknown CRC)
- Not sure yet if we got everything right

Field	Length (Byte)	Comment
Constant	1	0x70, 0x50
Serial Number	4	unique to device
Encrypted Data	16	
Checksum	1	CRC (unknown)

Hardware Analysis



Hardware Components

- SX1230 --> Radio Chip with public datasheet

The image shows the cover page of the SX1230 Datasheet. At the top left is the SEMTECH logo, which consists of a stylized 'S' icon followed by the word 'SEMTECH' in a bold, sans-serif font. To the right of the logo, the model number 'SX1230' is displayed in a large, bold, sans-serif font. Below the logo and model number, there are two horizontal bars: the first is dark blue with the text 'ADVANCED COMMUNICATIONS & SENSING' in white, and the second is white with the text 'DATASHEET' in dark blue. Underneath these bars, the title 'SX1230 - Integrated Transmitter IC' is written in bold, followed by a subtitle 'Narrow/wideband 315 MHz, 434 MHz 868 MHz and 915 MHz band Transmitter' which is underlined. The page is divided into two columns by a vertical line. The left column has a dark blue header 'GENERAL DESCRIPTION' and contains the text: 'The SX1230 is a fully integrated transmitter which can operate in the 315, 434, 868 and 915 MHz licence free ISM bands.' The right column has a dark blue header 'APPLICATIONS' and contains a bulleted list: '◆ Remote Keyless Entry (RKE)' and '◆ Remote Control / Security Systems'.

Hardware Components

- Unknown microcontroller (PIC ?)
- Microchip Logo and „EE001 20000 1507AHW“
- What do we know ?
 - 28 Pins
 - QFN package
 - Chip older than 3 years, so no Atmel
 - GND on Pins 5 and 16
 - PCB likely has test points for programming

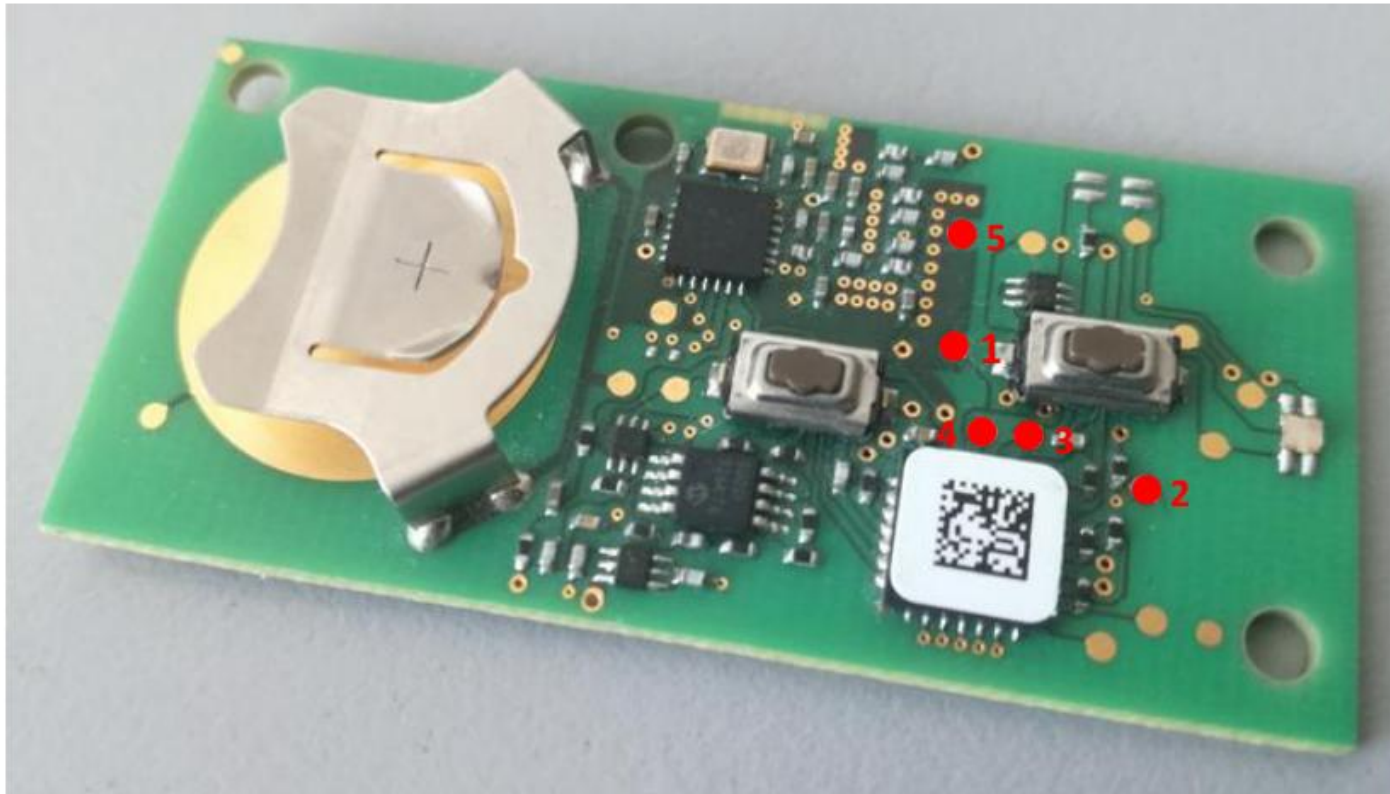
Identifying the Controller

- Let's search a component distributor:
 - “manufacturer: Microchip”, 28-pins, QFN
 - Results: PIC16F, PIC18F, PIC24F, PIC32F
 - Let's look into their datasheets !
- PIC24 and PIC32 would need GND on Pin24, we don't have that on the PCB
--> It must be either PIC16F or PIC18F

Programming Pins

- Pins required for PIC programming are: /MCLR, PGD, PGC as well as Vdd and GND
- 28-pin PIC16F and PIC18F have their programming pins at the same location
- Let's PICkit3 it to read the device number !

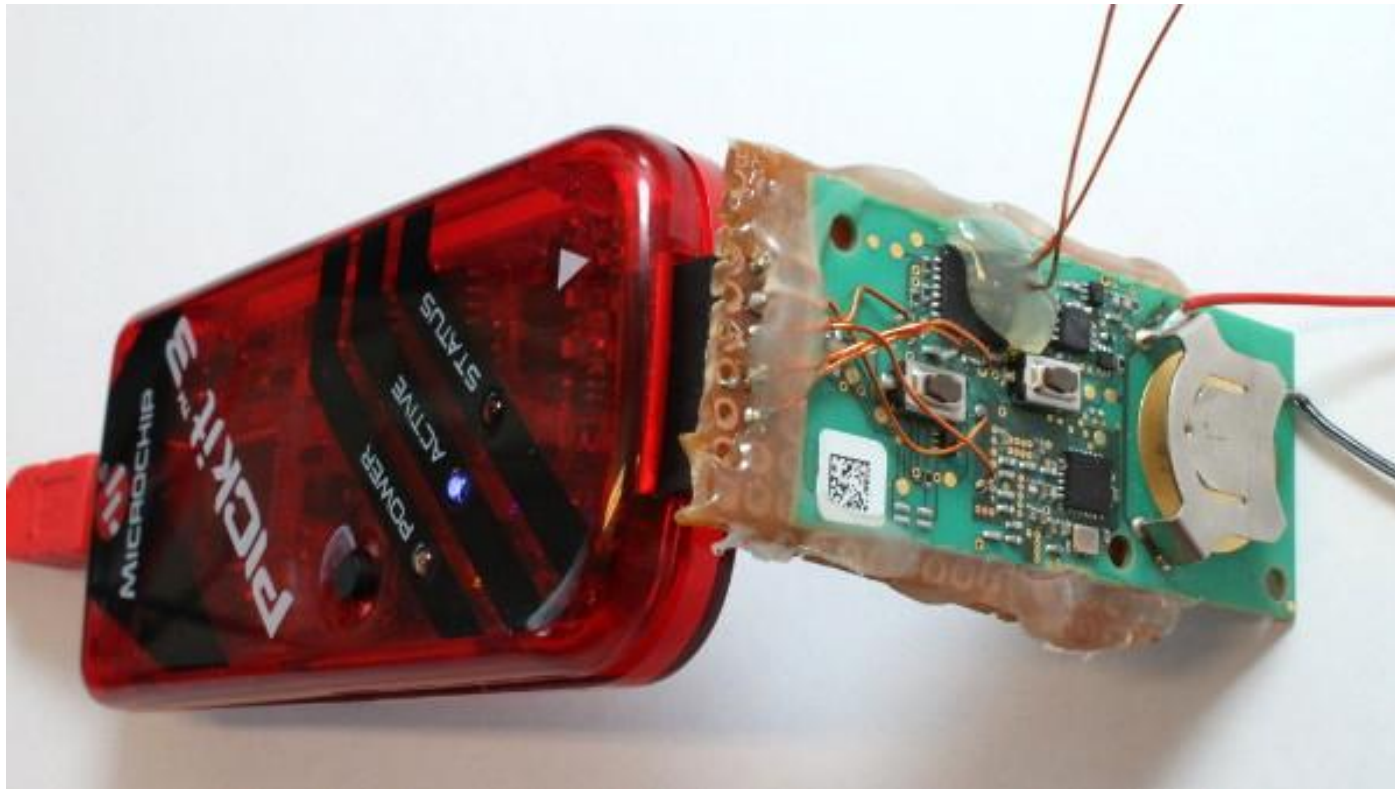
Programming Pads



1	$\overline{\text{MCLR}}$
2	V _{DD}
3	V _{SS}
4	PGD
5	PGC

PICkit3

- We created a little break-out board



PICkit3: Results

- MCU is PIC18F26K20

The screenshot shows the PICkit 3 Programmer interface for a PIC18F26K20 device. The configuration window displays the following details:

- Device: PIC18F26K20
- Configuration: 0800 0F1D 0100 0081
- User IDs: FF FF FF FF FF FF FF FF
- Code Protect: 8000 E00F 400F
- Checksum: 0A6C
- OSCCAL: BandGap:

The status bar indicates "Reading device: Program Memory... EE... UserIDs... Config... Done." and shows a VDD PICkit 3 level of 3.3V.

The Program Memory section is enabled and shows a source of "Read from PIC18F26K20". The memory data is displayed in a table:

Address	0000	0010	0020	0030	0040	0050	0060	0070	0080	0090	00A0	00B0
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0010	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0020	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0030	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0040	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0050	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0060	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0070	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0080	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0090	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
00A0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
00B0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000

The EEPROM Data section is also enabled and shows the following data:

Address	000	010	020	030
000	FF FF FF FF FF A0 4E 00 00 FF 00 00 01 F0 F2			
010	81 86 F0 79 E0 B0 87 B1 31 F1 43 2C D1 96 F2 30			
020	4F 9C 27 2E F1 E7 62 2B 42 86 04 3F 3D 68 04 01			
030	00 FF 00 00 02 D6 FF FF FF FF FF 00 00 01 F0 F2			

The interface also includes buttons for Read, Write, Verify, Erase, and Blank Check, and a VDD PICkit 3 control section with a 3.3V voltage selector.

PICkit3: Results

- All Flash blocks are locked (code protection)

Configuration Word Editor

Device Configuration Words may be edited here at the bit level. Refer to device datasheet for specific configuration bit functions.

= Unimplemented bit = Configuration bit. Click to toggle value.

Name	Address	Value	Bit Edit															
			15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CONFIG1	300000	0800	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CONFIG2	300002	0F1D	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
CONFIG3	300004	0100	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CONFIG4	300006	0081	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
CONFIG5	300008	8000	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
CONFIG6	30000A	E00F	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
CONFIG7	30000C	400F	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Unimplemented bits are displayed in the Value column as selected in menu *Tools > Display Unimplemented Config Bits*

Save Cancel

EEPROM
not locked

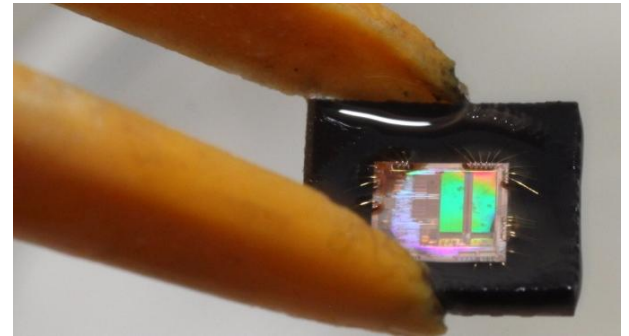
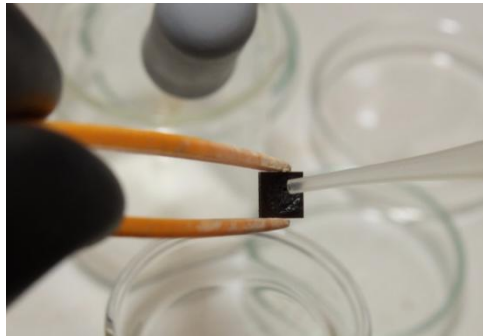
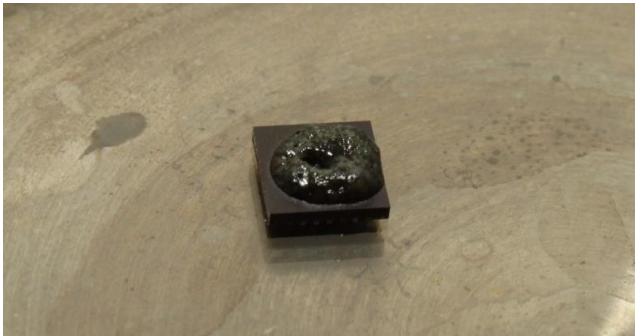
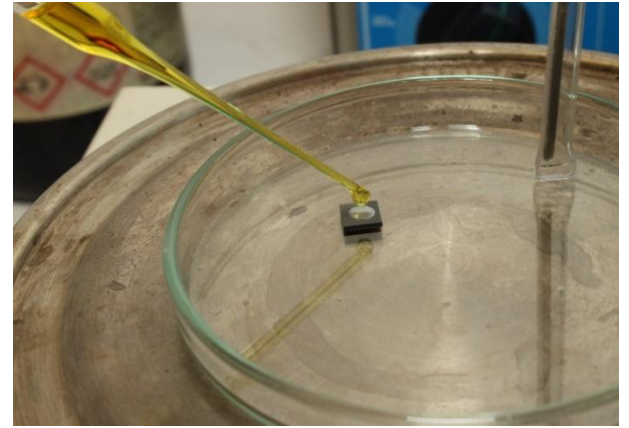
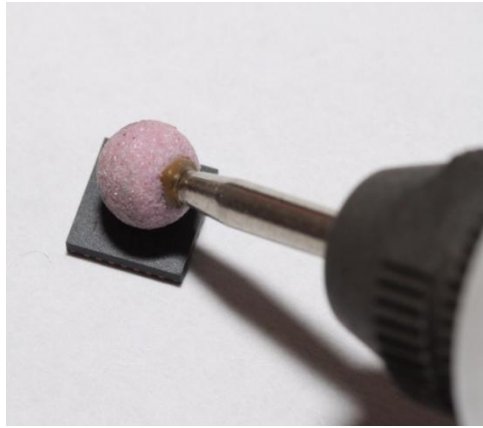
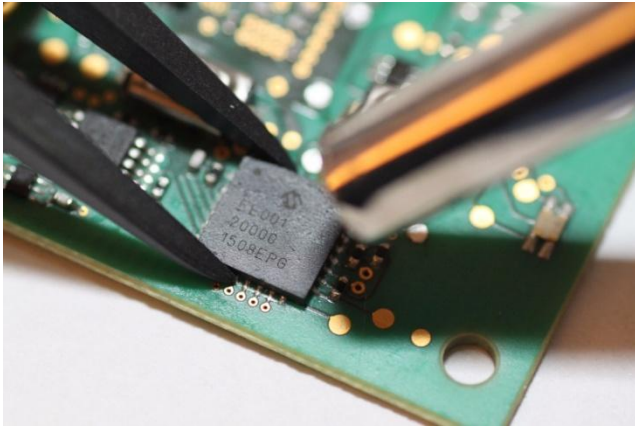
PIC Firmware Extraction Techniques

- To analyze the firmware and thus the system for security vulnerabilities, we need to analyze the firmware
- The firmware is currently locked
- PIC locking mechanism:
 - “locking bits” --> security fuse
- Back to security-by-obscurity ?

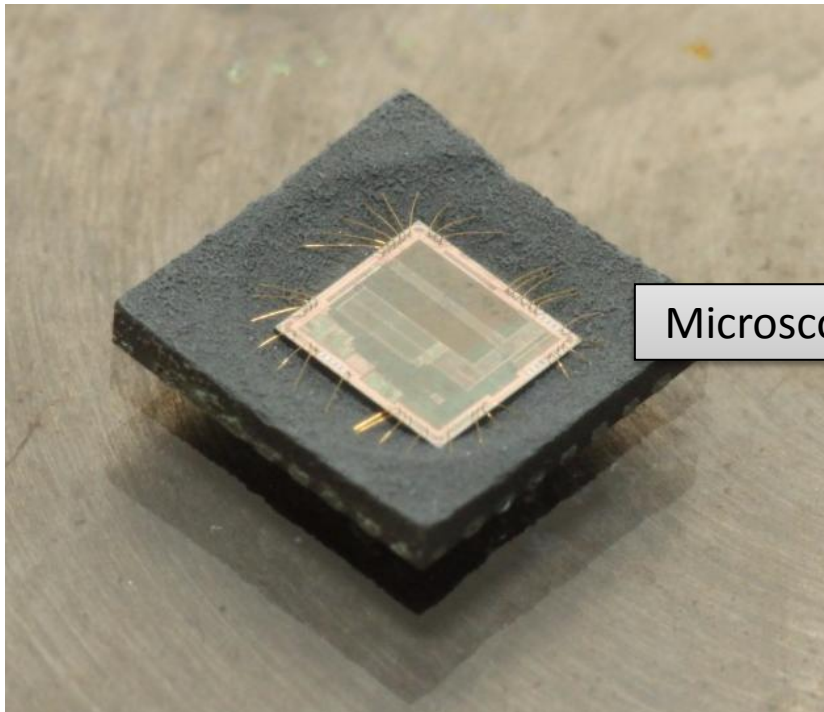
IC Analysis

- How is the security fuse logic implemented on the PIC18F microcontroller ?
- Approach:
 - IC Decapsulation
 - (Rough) microscopic analysis

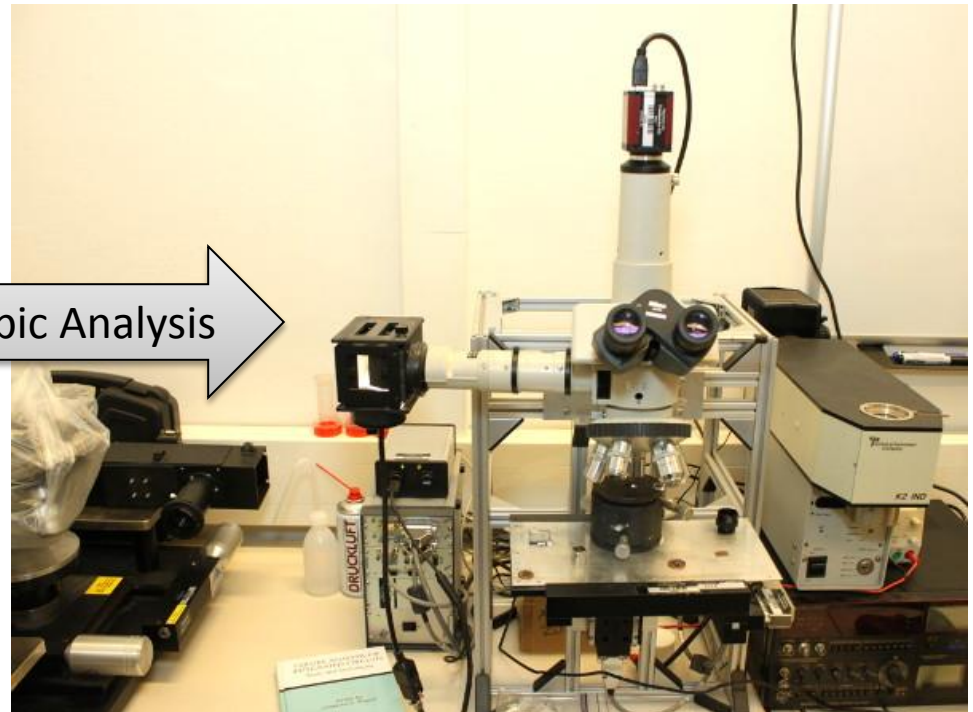
IC Analysis



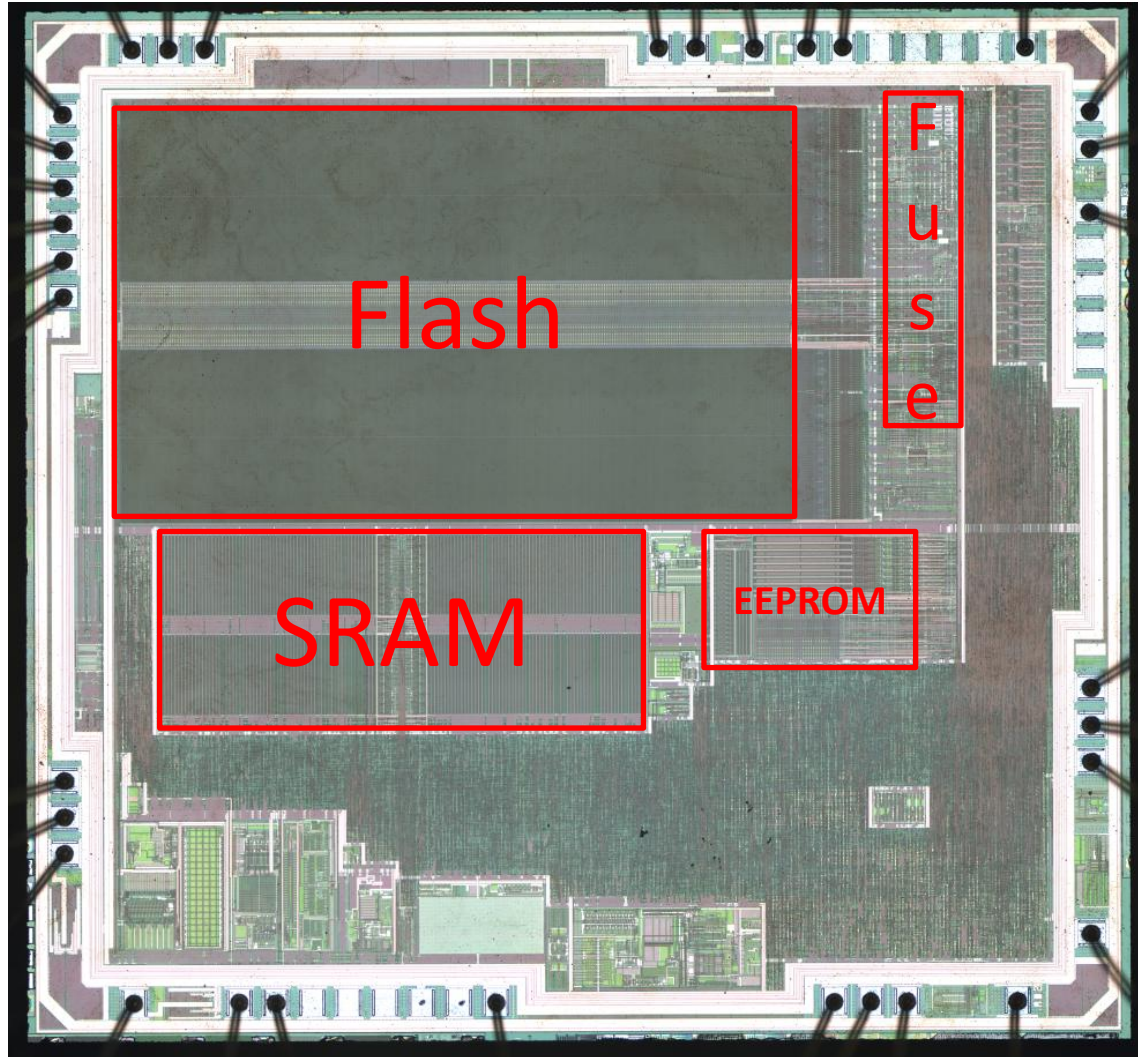
IC Analysis



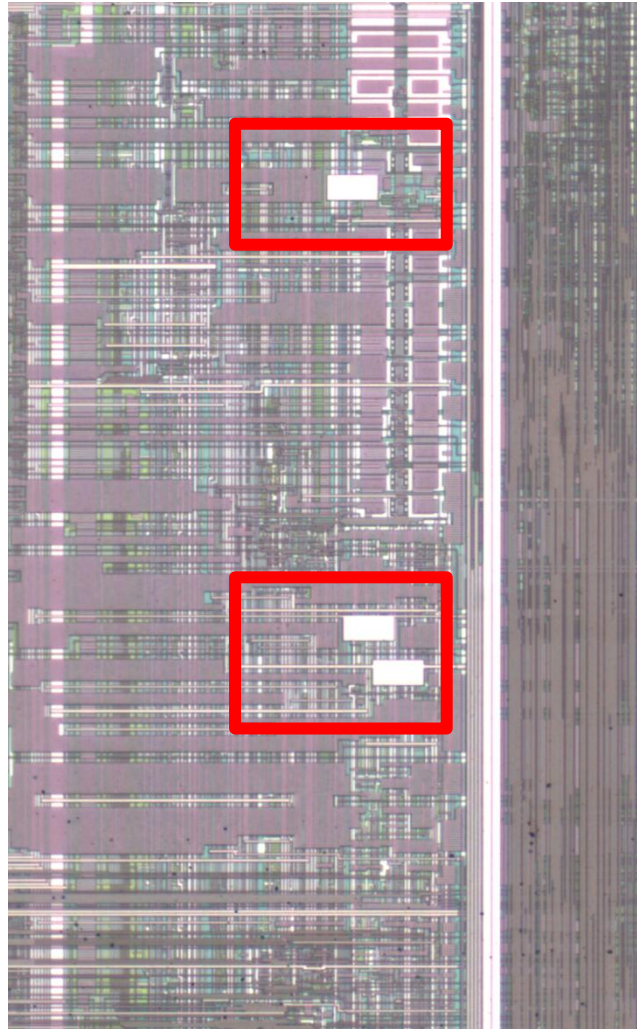
Microscopic Analysis



IC Analysis



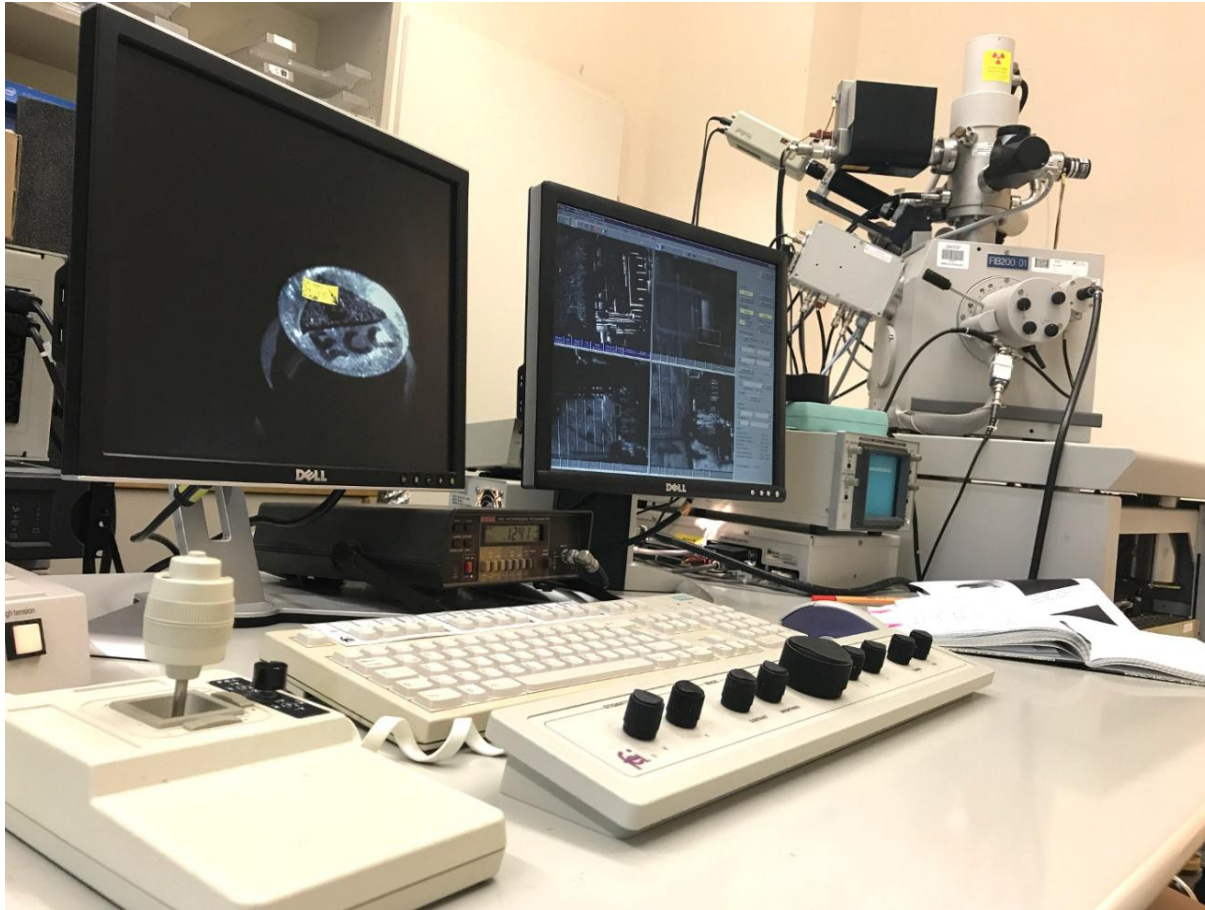
Fuse Close-up



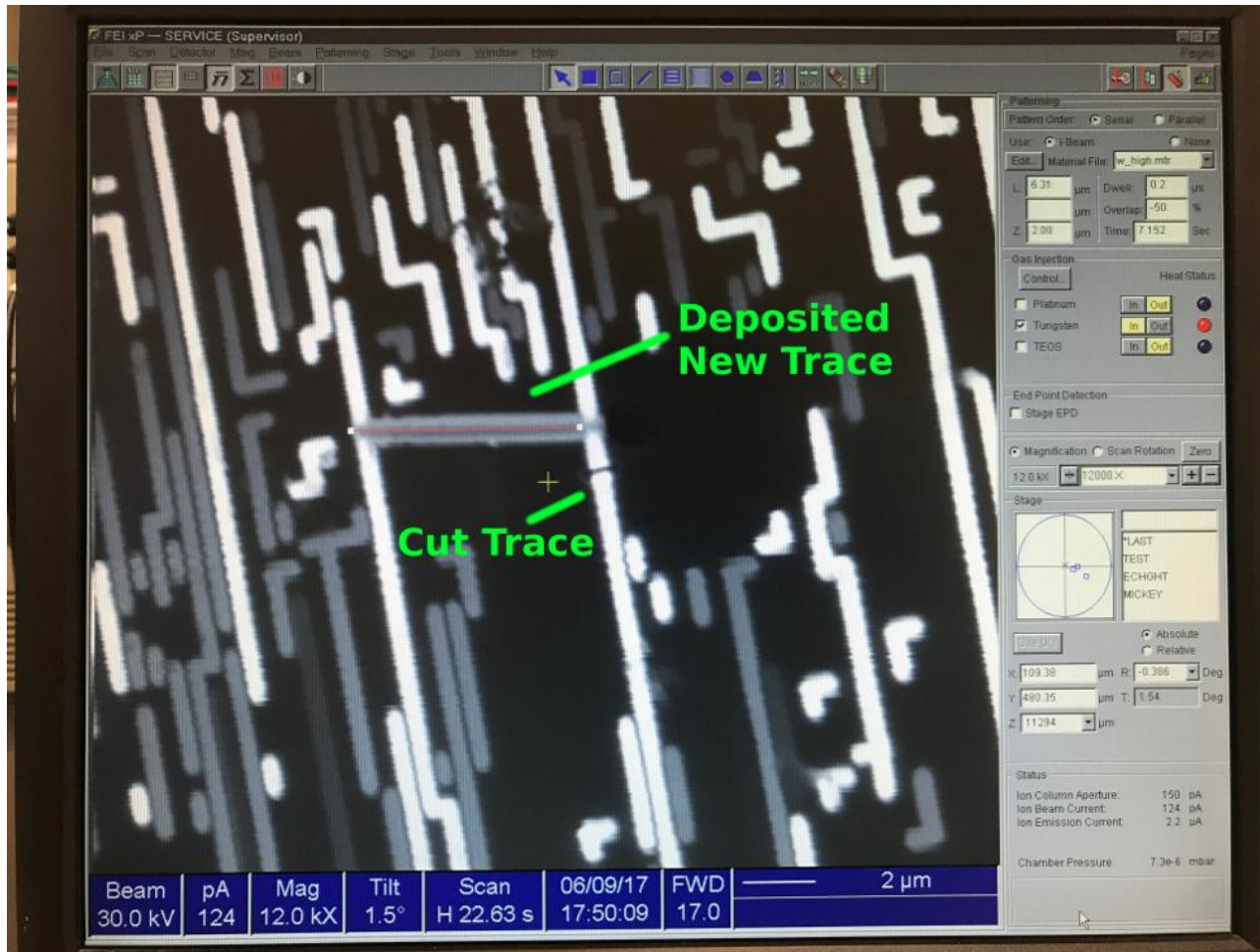
Method 1: Invasive FIB Circuit Edit

- Approach:
 1. Identify security fuse logic (might involve IC deprocessing and analysis other PICs)
 2. Bypass security fuse with FIB circuit edit
- Advantage: High success rate
- Disadvantage: Large time effort if no existing recipe

Method 1: Invasive FIB Circuit Edit



Method 1: Invasive FIB Circuit Edit

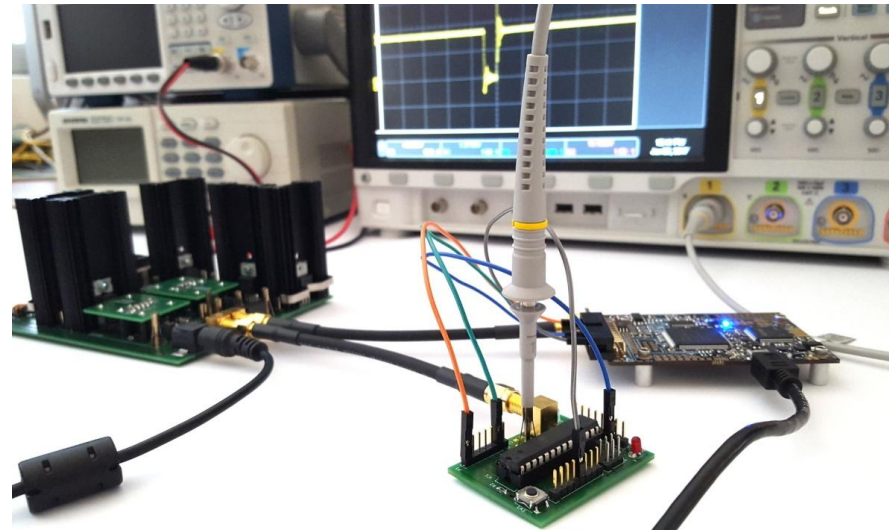


Method 2: „Bunnie“ Attack

- Fuses can be cleared with UV, but Flash needs to be protected (e.g. tape or permanent marker)
- Shields avoid direct UV exposure (see our die image)
- UV light might still get in at a steep angle
--> “Bunnie” attack (original idea by Bunnie Huang)
- Advantage: Easy to conduct and our chip is already open
- Disadvantage: Unclear whether we have anti-fuses, potential to damage bond-wires during masking

Method 3: Voltage Glitching

- Our glitch amplifier can deliver $<12\text{ns}$ pulses at high drive current



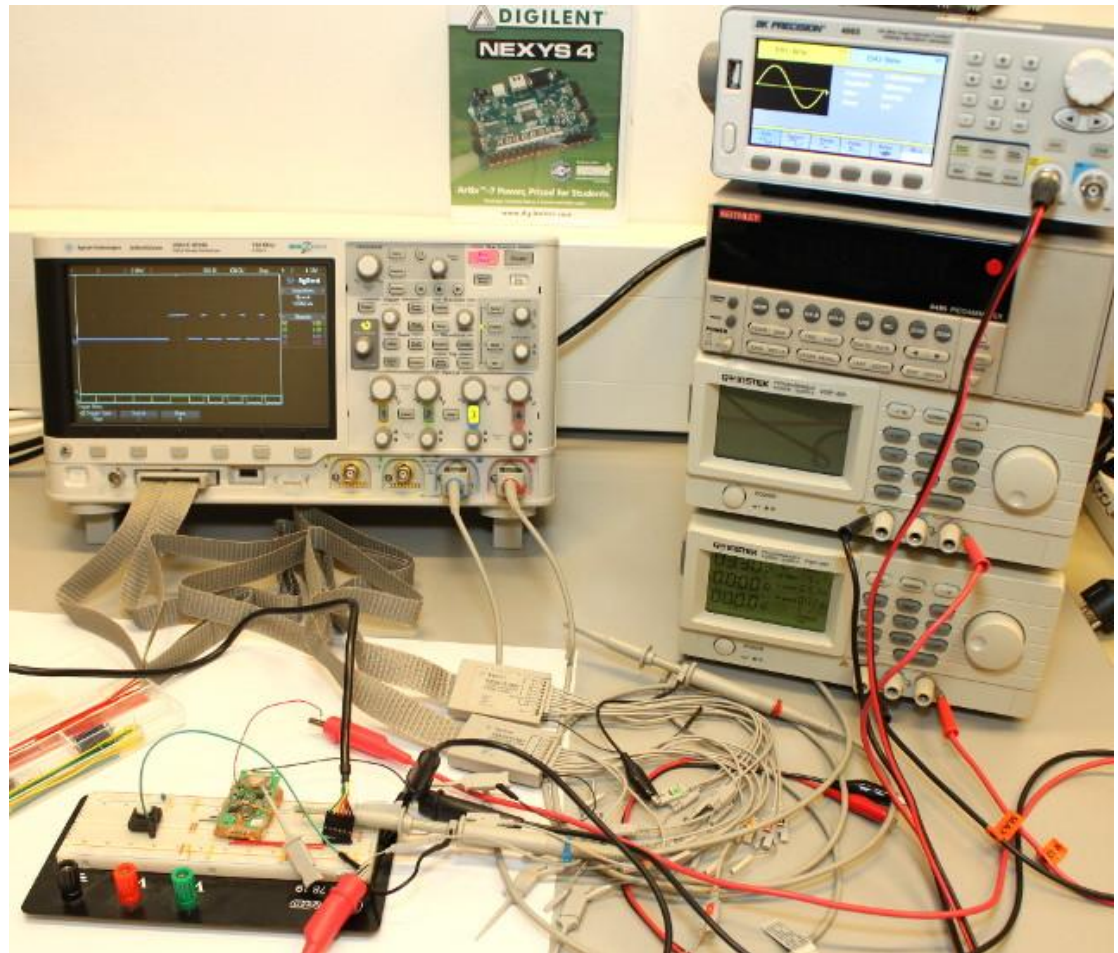
- Advantage: easy to conduct
- Disadvantage: Unclear if successful on PIC18F, glitch parameters need to be found first

Method 4: The Easy Way :-)

- There is a trivial design issue in the PIC18F security fuse logic
- Memory blocks have individual fuse bits
- A block can be reprogrammed while the others keep their original content
- Presented at 27C3 (2010):
 - [Milosch Meriac, 27C3: Heart of Darkness – exploring the uncharted backwaters of HID iCLASS security, <https://www.openpcd.org/images/HID-iCLASS-security.pdf>]

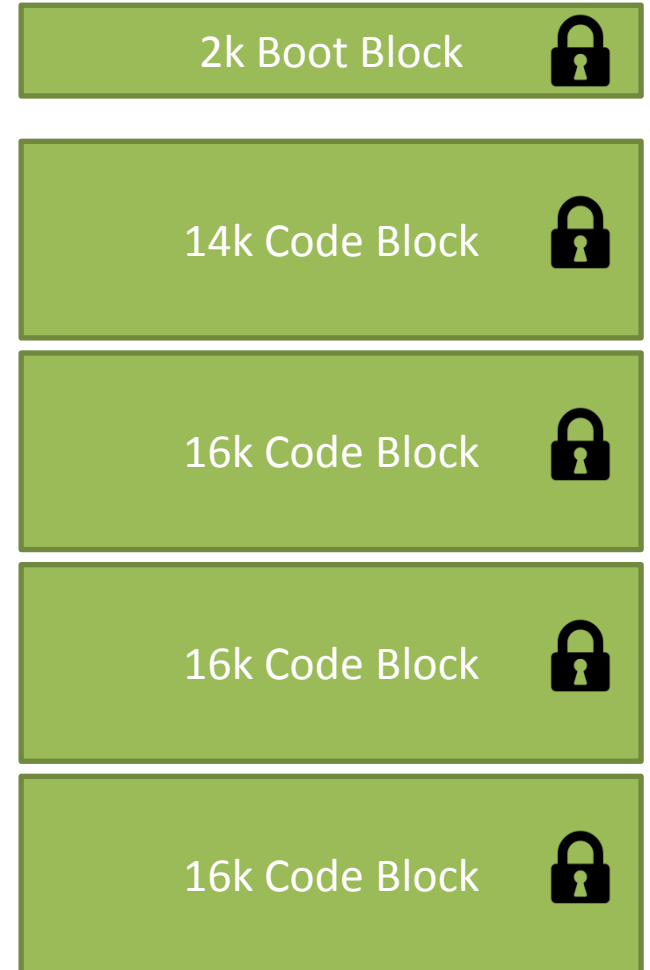
We Implemented a PIC Firmware Extraction Tool

- Breadboard HW
- FTDI (USB-TTL)
- Short SW Script



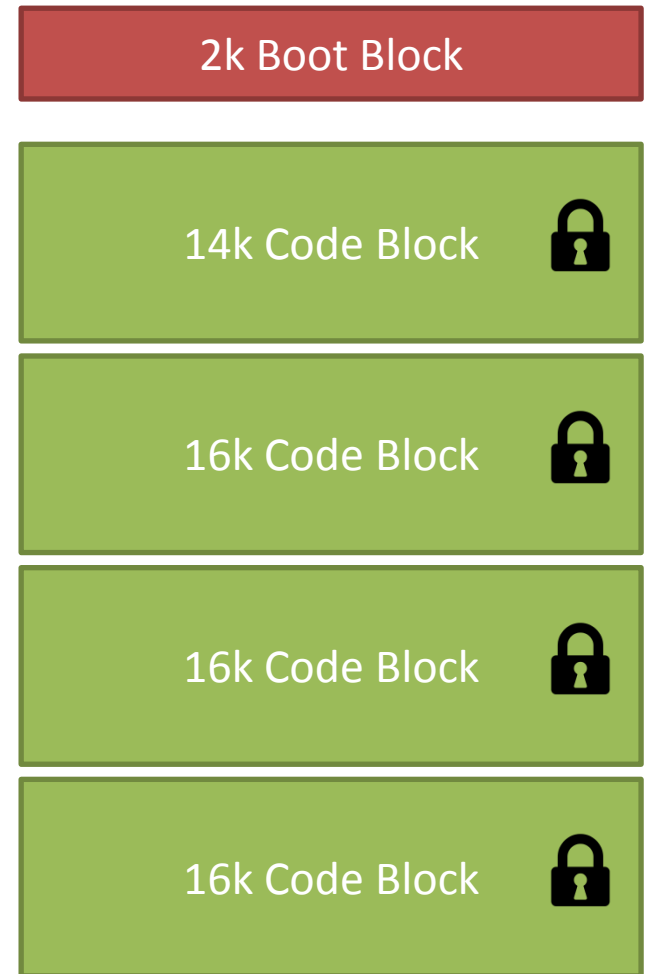
PIC Firmware Extraction

- 5 memory blocks (Flash)
- Individual lock bits

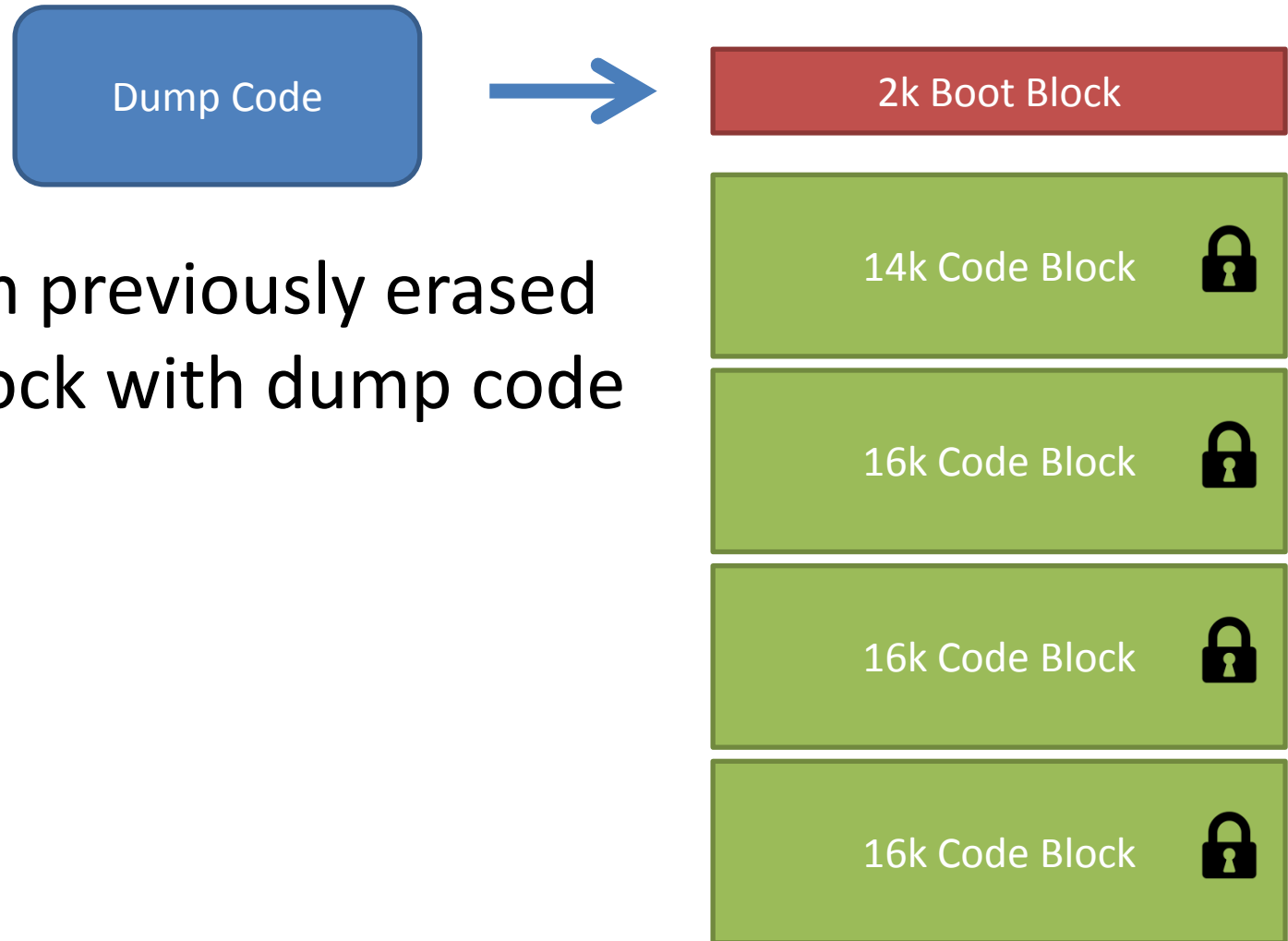


PIC Firmware Extraction

- Remove lock bit of boot block --> boot block is erased
- The other blocks remain intact



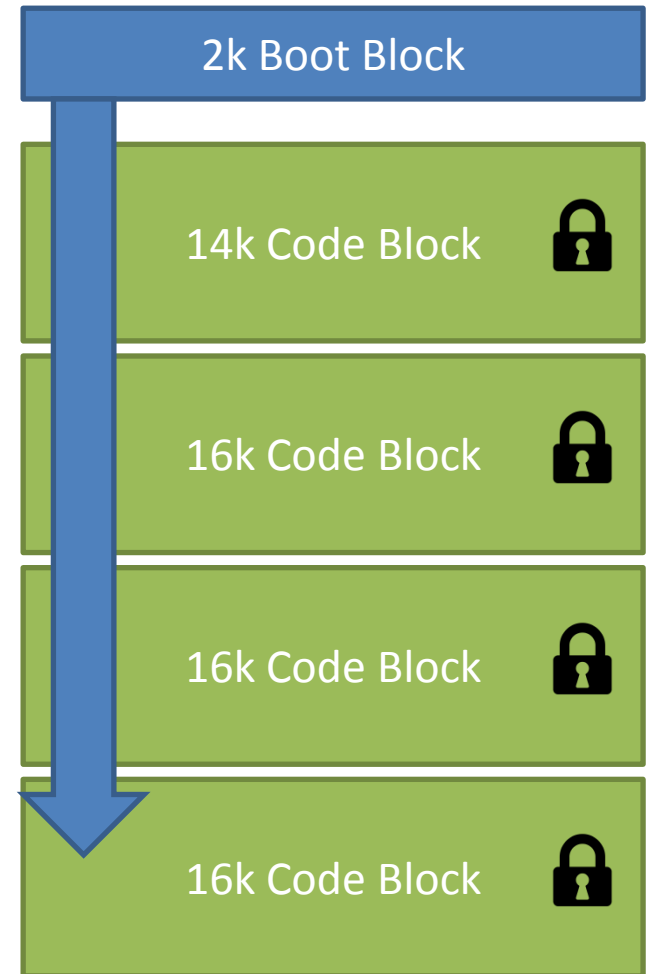
PIC Firmware Extraction



- Program previously erased boot block with dump code

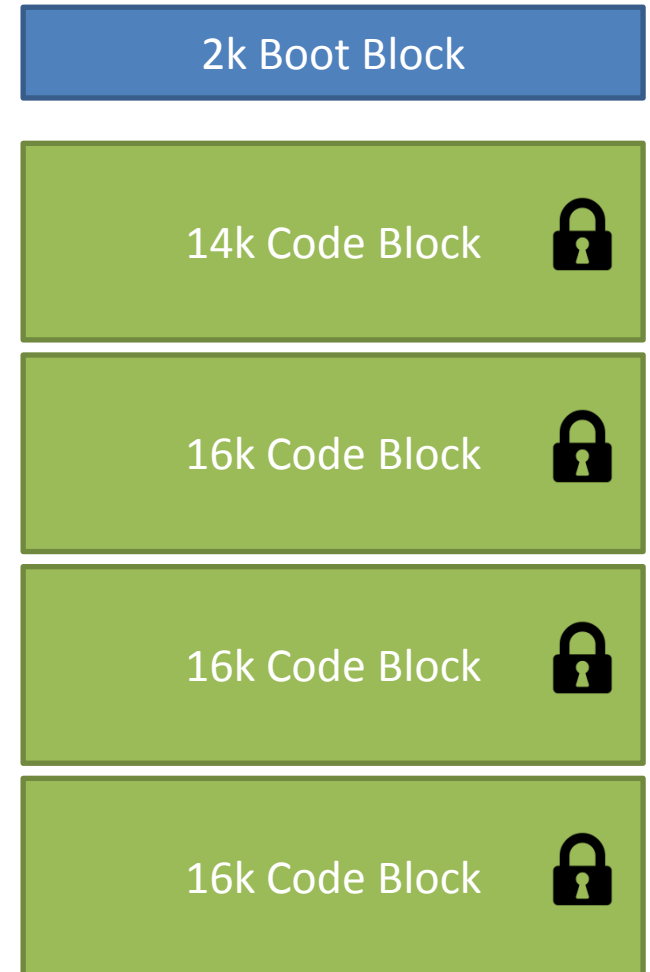
PIC Firmware Extraction

- Dump code reads remaining 4 code blocks
- Output on UART



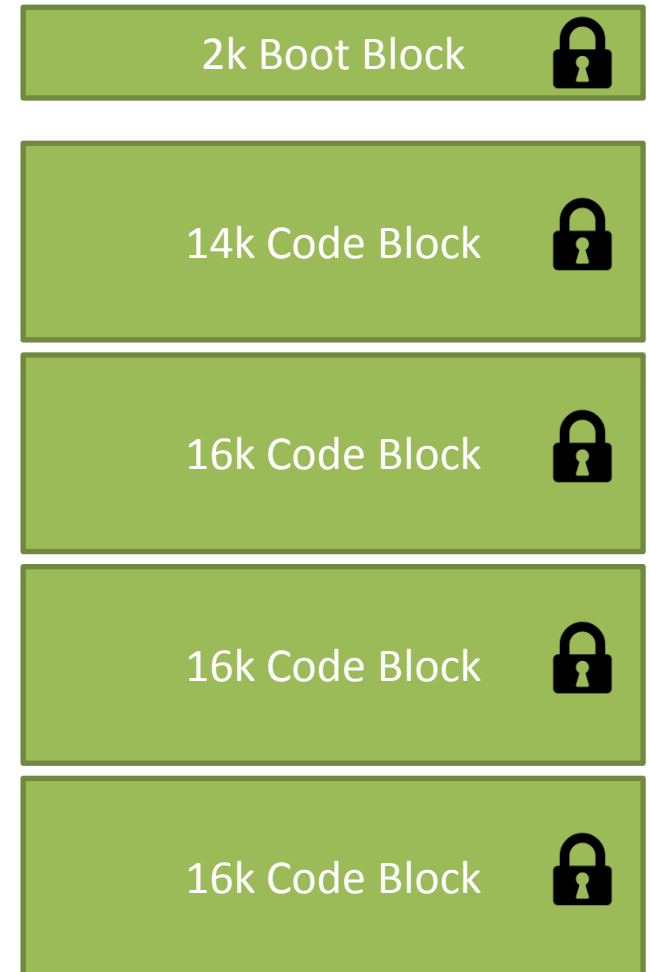
PIC Firmware Extraction

- We have a successful dump of the 4 code blocks now
- We overwrote the 2k boot block and the content is lost :-)



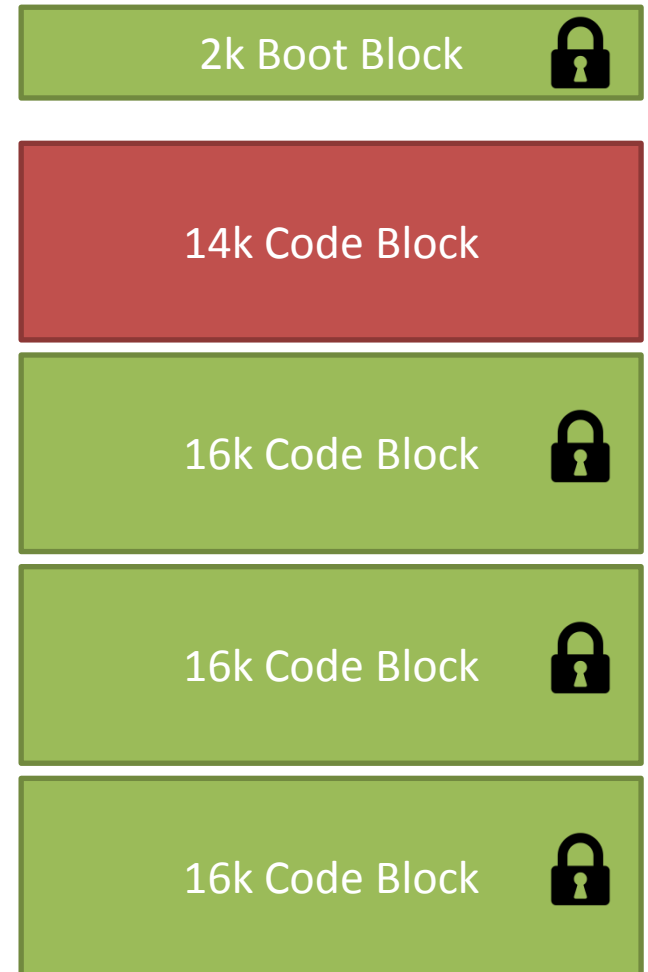
PIC Firmware Extraction

- Solution: Take another PIC with identical programming
- Restart the process

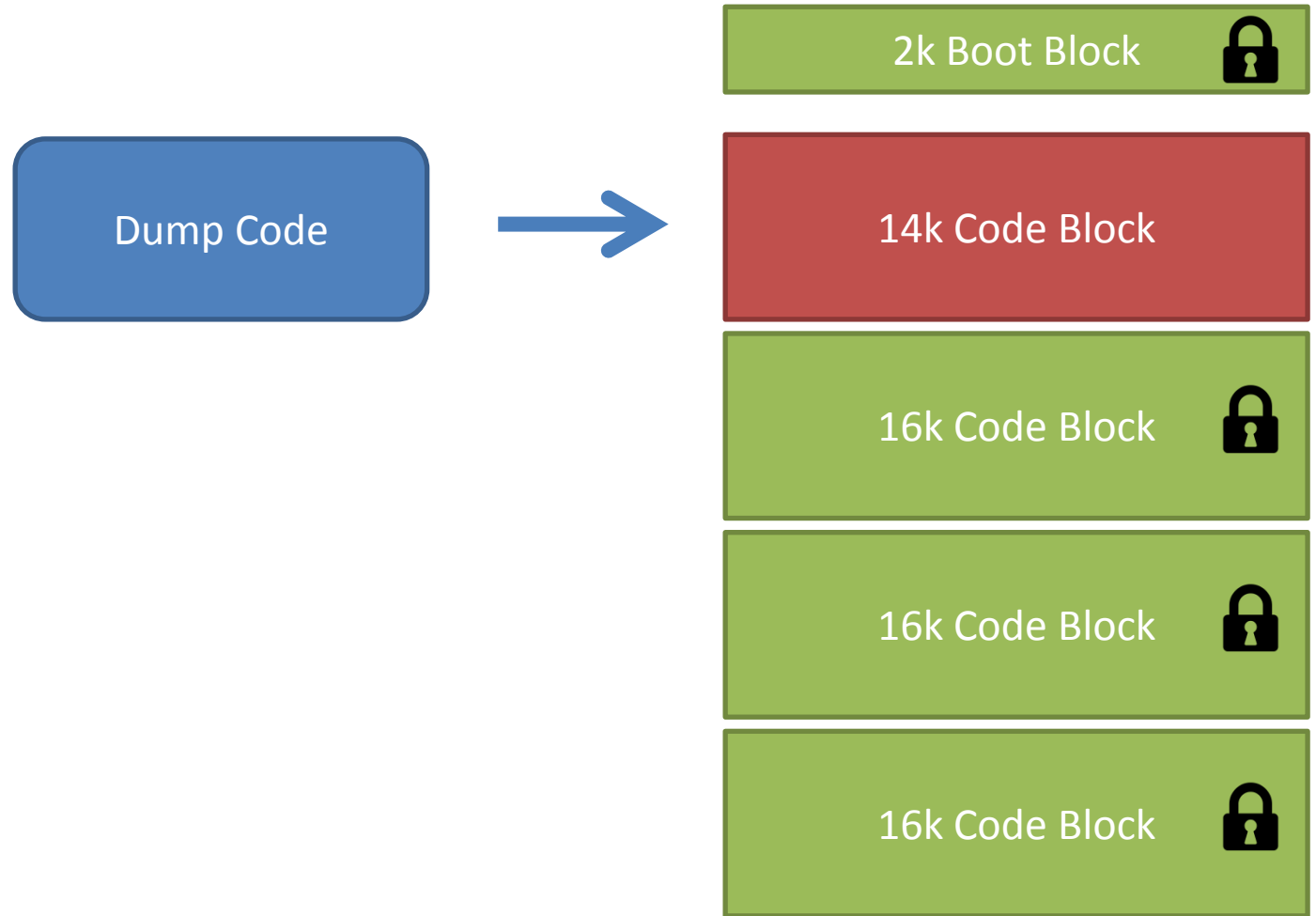


PIC Firmware Extraction

- This time we unlock a code block
- The code block is thus erased

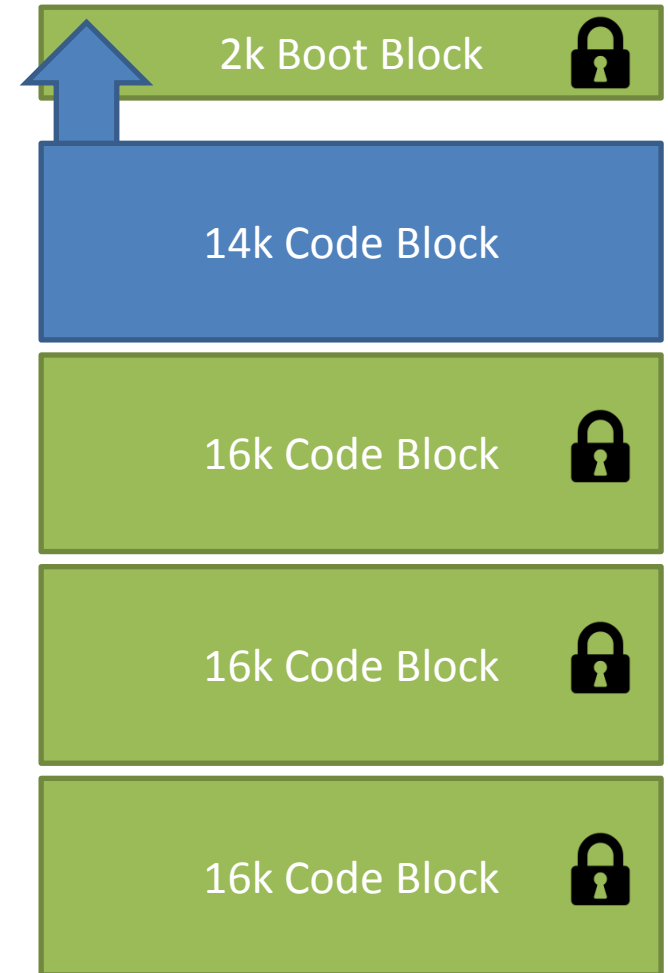


PIC Firmware Extraction



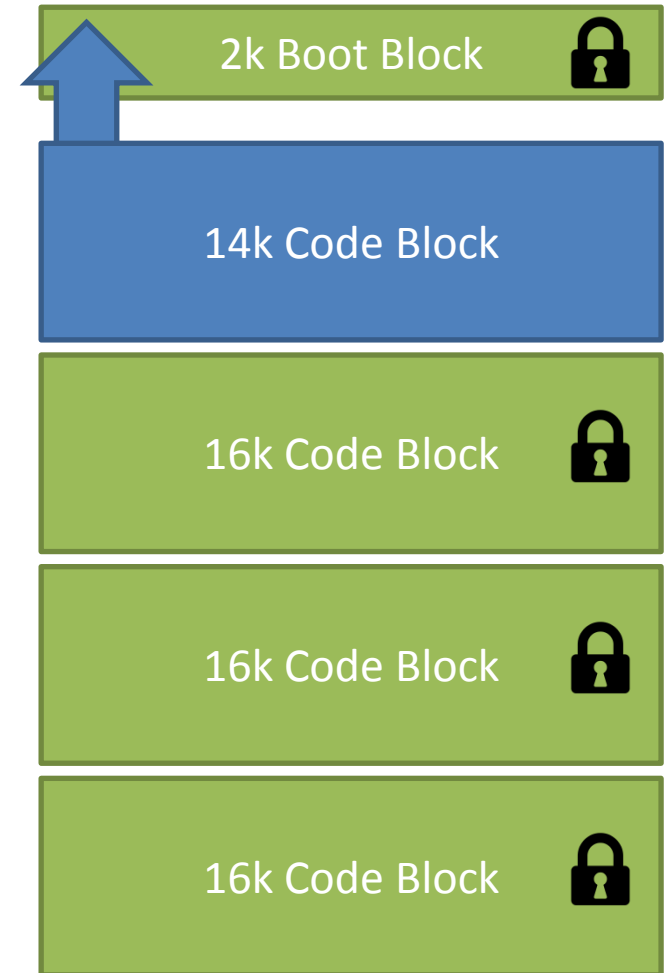
PIC Firmware Extraction

- Wait until execution jumps from another block to our dump block
- NOP slide
- Read the boot block



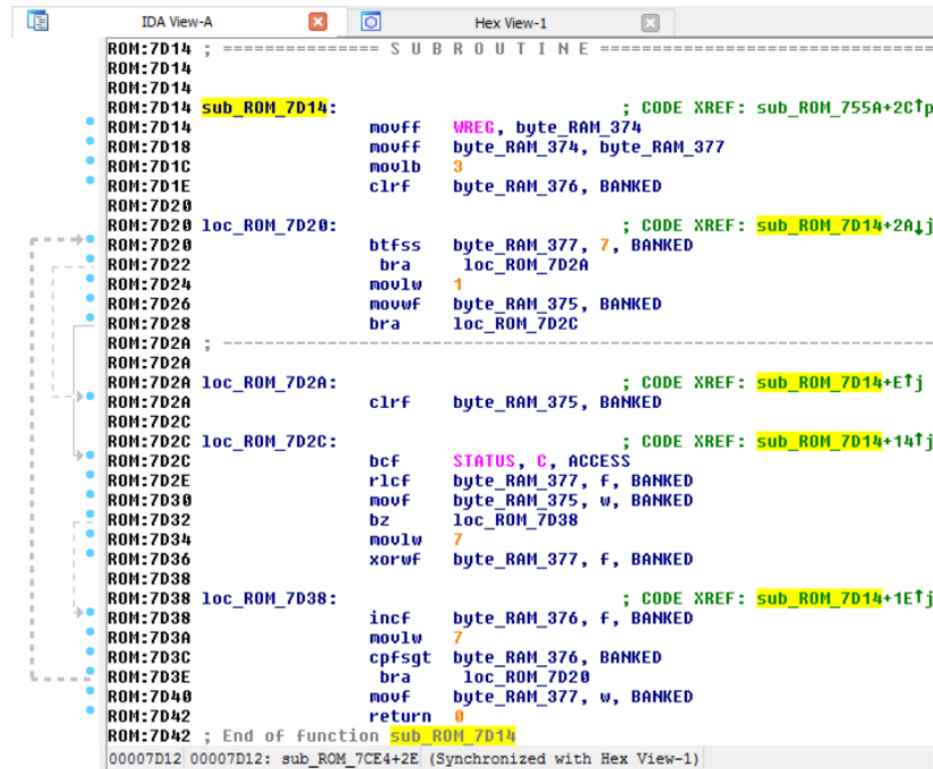
PIC Firmware Extraction

- Both PICs are (partially) dumped now
- We combine the two partials dumps to obtain a full dump
- We can re-flash the two PICs so that they contain their original programming



Firmware Analysis

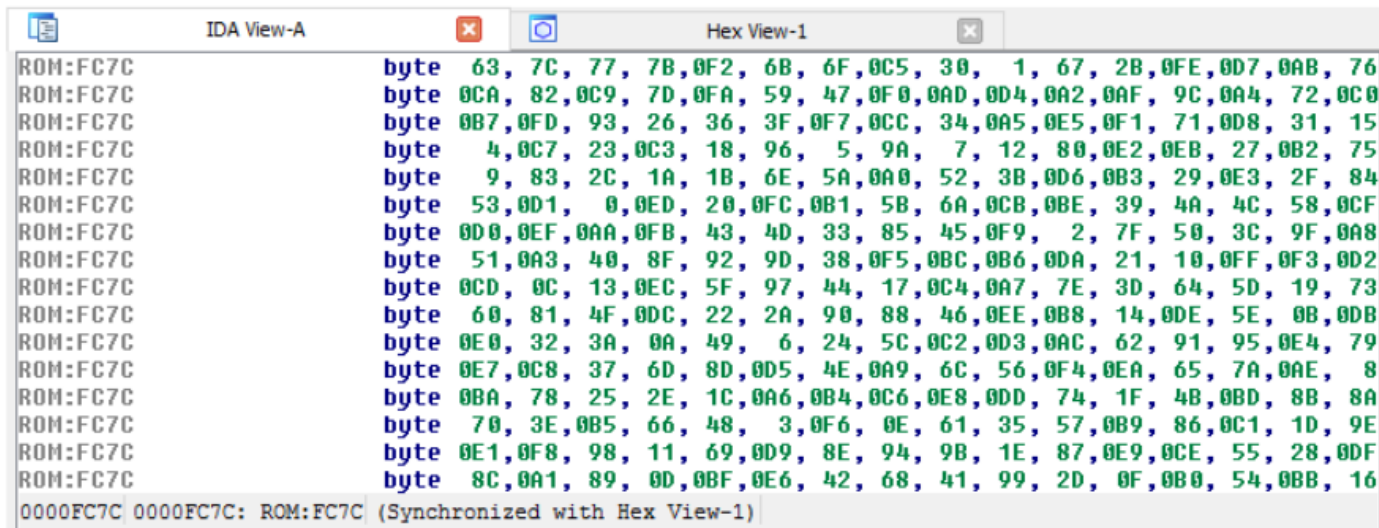
- Analysis with Ida Pro tool (supports PIC)



```
ROM:7D14 ; ----- S U B R O U T I N E -----
ROM:7D14
ROM:7D14
ROM:7D14 sub_ROM_7D14: ; CODE XREF: sub_ROM_755A+2C1p
ROM:7D14     movff   WREG, byte_RAM_374
ROM:7D18     movff   byte_RAM_374, byte_RAM_377
ROM:7D1C     movlb   3
ROM:7D1E     clrf   byte_RAM_376, BANKED
ROM:7D20
ROM:7D20 loc_ROM_7D20: ; CODE XREF: sub_ROM_7D14+2A1j
ROM:7D20     btfss  byte_RAM_377, 7, BANKED
ROM:7D22     bra    loc_ROM_7D2A
ROM:7D24     movlw  1
ROM:7D26     movwf  byte_RAM_375, BANKED
ROM:7D28     bra    loc_ROM_7D2C
ROM:7D2A ; -----
ROM:7D2A loc_ROM_7D2A: ; CODE XREF: sub_ROM_7D14+E1fj
ROM:7D2A     clrf   byte_RAM_375, BANKED
ROM:7D2C
ROM:7D2C loc_ROM_7D2C: ; CODE XREF: sub_ROM_7D14+141j
ROM:7D2C     bcf   STATUS, C, ACCESS
ROM:7D2E     rlc   byte_RAM_377, f, BANKED
ROM:7D30     movf  byte_RAM_375, w, BANKED
ROM:7D32     bz    loc_ROM_7D38
ROM:7D34     movlw 7
ROM:7D36     xorwf byte_RAM_377, f, BANKED
ROM:7D38
ROM:7D38 loc_ROM_7D38: ; CODE XREF: sub_ROM_7D14+1E1j
ROM:7D38     incf  byte_RAM_376, f, BANKED
ROM:7D3A     movlw 7
ROM:7D3C     cpfsgt byte_RAM_376, BANKED
ROM:7D3E     bra    loc_ROM_7D20
ROM:7D40     movf  byte_RAM_377, w, BANKED
ROM:7D42     return 0
ROM:7D42 ; End of function sub_ROM_7D14
00007D12 00007D12: sub_ROM_7CE4+2E (Synchronized with Hex View-1)
```

Firmware Analysis

- Firmware analysis provides deep implementation insight, including: RF protocol, checksum computation, encryption methods



The screenshot shows the IDA Pro Hex View window. The left pane displays assembly instructions for ROM:FC7C, and the right pane shows the corresponding hex values. The instructions are 'byte' instructions, each followed by a list of 16 hex values. The hex values are displayed in green text.

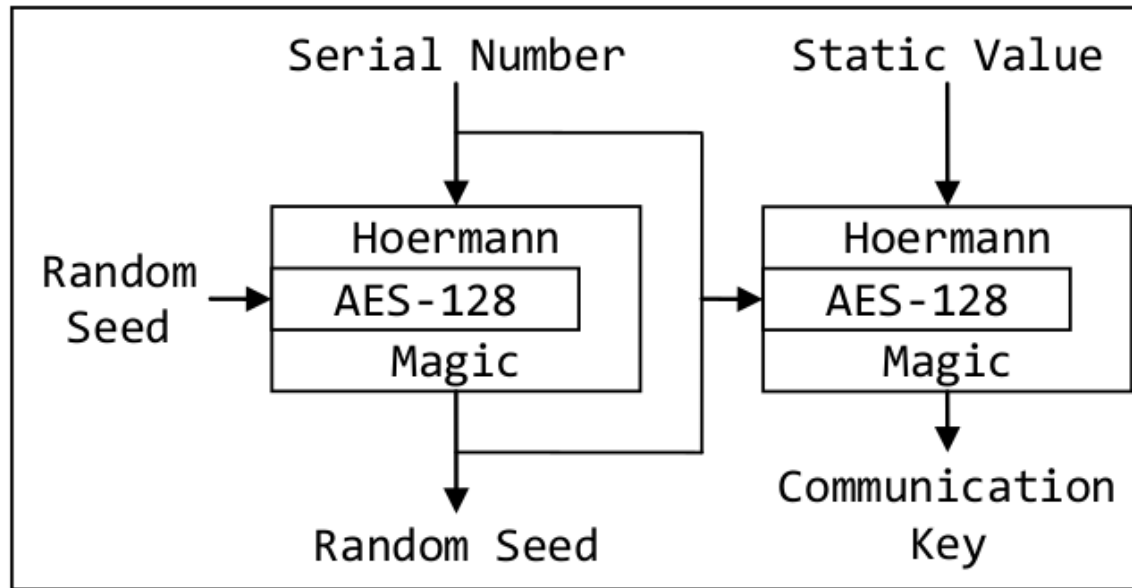
```
ROM:FC7C byte 63, 7C, 77, 7B, 0F2, 6B, 6F, 0C5, 30, 1, 67, 2B, 0FE, 0D7, 0AB, 76
ROM:FC7C byte 0CA, 82, 0C9, 7D, 0FA, 59, 47, 0F0, 0AD, 0D4, 0A2, 0AF, 9C, 0A4, 72, 0C0
ROM:FC7C byte 0B7, 0FD, 93, 26, 36, 3F, 0F7, 0CC, 34, 0A5, 0E5, 0F1, 71, 0D8, 31, 15
ROM:FC7C byte 4, 0C7, 23, 0C3, 18, 96, 5, 9A, 7, 12, 80, 0E2, 0EB, 27, 0B2, 75
ROM:FC7C byte 9, 83, 2C, 1A, 1B, 6E, 5A, 0A0, 52, 3B, 0D6, 0B3, 29, 0E3, 2F, 84
ROM:FC7C byte 53, 0D1, 0, 0ED, 20, 0FC, 0B1, 5B, 6A, 0CB, 0BE, 39, 4A, 4C, 58, 0CF
ROM:FC7C byte 0D0, 0EF, 0AA, 0FB, 43, 4D, 33, 85, 45, 0F9, 2, 7F, 50, 3C, 9F, 0A8
ROM:FC7C byte 51, 0A3, 40, 8F, 92, 9D, 38, 0F5, 0BC, 0B6, 0DA, 21, 10, 0FF, 0F3, 0D2
ROM:FC7C byte 0CD, 0C, 13, 0EC, 5F, 97, 44, 17, 0C4, 0A7, 7E, 3D, 64, 5D, 19, 73
ROM:FC7C byte 60, 81, 4F, 0DC, 22, 2A, 90, 88, 46, 0EE, 0B8, 14, 0DE, 5E, 0B, 0DB
ROM:FC7C byte 0E0, 32, 3A, 0A, 49, 6, 24, 5C, 0C2, 0D3, 0AC, 62, 91, 95, 0E4, 79
ROM:FC7C byte 0E7, 0C8, 37, 6D, 8D, 0D5, 4E, 0A9, 6C, 56, 0F4, 0EA, 65, 7A, 0AE, 8
ROM:FC7C byte 0BA, 78, 25, 2E, 1C, 0A6, 0B4, 0C6, 0E8, 0DD, 74, 1F, 4B, 0BD, 8B, 8A
ROM:FC7C byte 70, 3E, 0B5, 66, 48, 3, 0F6, 0E, 61, 35, 57, 0B9, 86, 0C1, 1D, 9E
ROM:FC7C byte 0E1, 0F8, 98, 11, 69, 0D9, 8E, 94, 9B, 1E, 87, 0E9, 0CE, 55, 28, 0DF
ROM:FC7C byte 8C, 0A1, 89, 0D, 0BF, 0E6, 42, 68, 41, 99, 2D, 0F, 0B0, 54, 0BB, 16
```

0000FC7C 0000FC7C: ROM:FC7C (Synchronized with Hex View-1)

RIJNDAEL S-BOX FOR AES

Firmware Analysis Result

- Cryptographic scheme and key generation

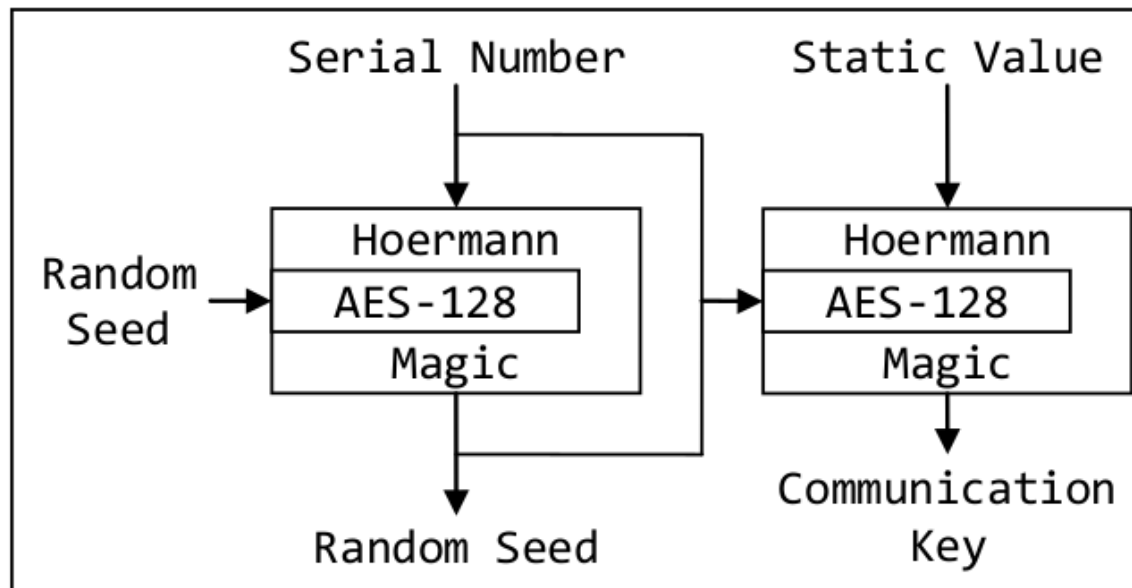


GENERATION OF COMMUNICATION KEYS

System Analysis

Field	Length (Byte)	Comment
Constant	1	0x70, 0x50
Serial Number	4	unique to device
Encrypted Data	16	adapted AES-128
Checksum	1	CRC

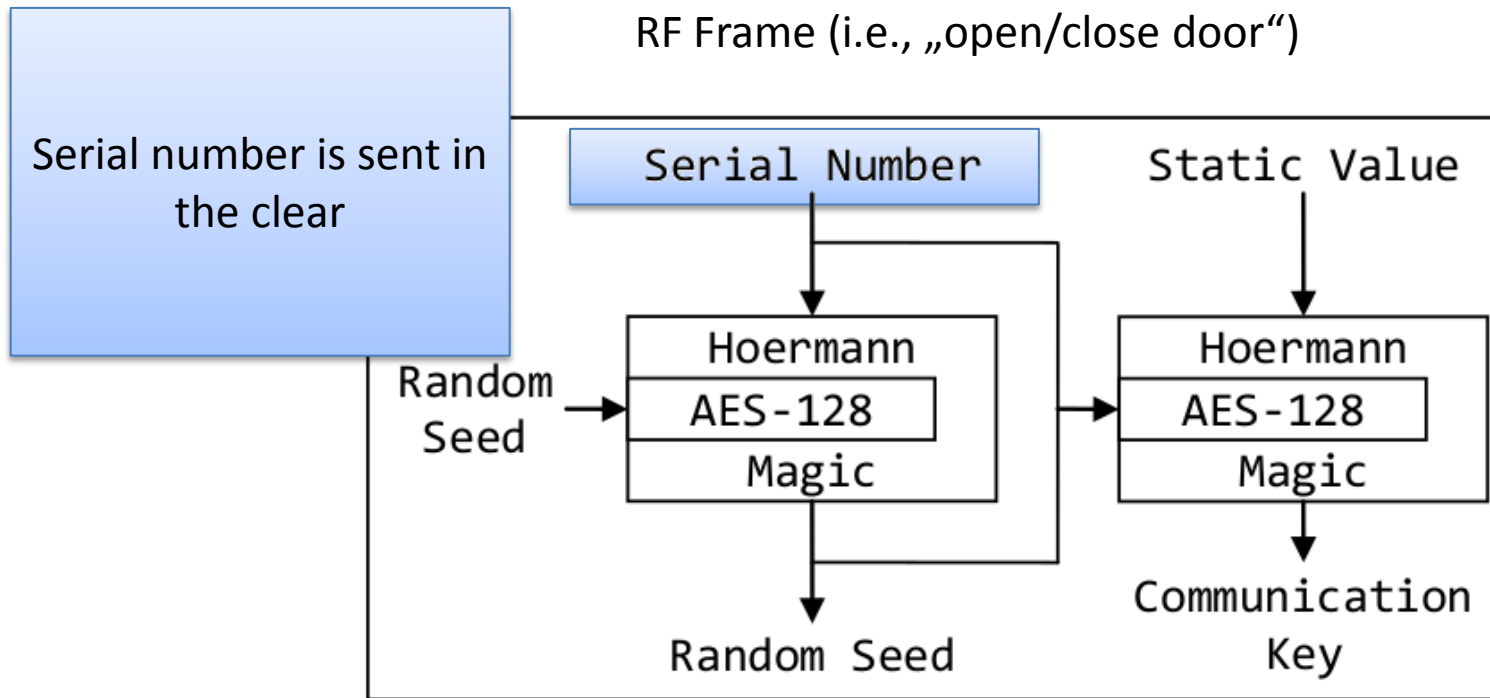
RF Frame (i.e., „open/close door“)



GENERATION OF COMMUNICATION KEYS

System Analysis

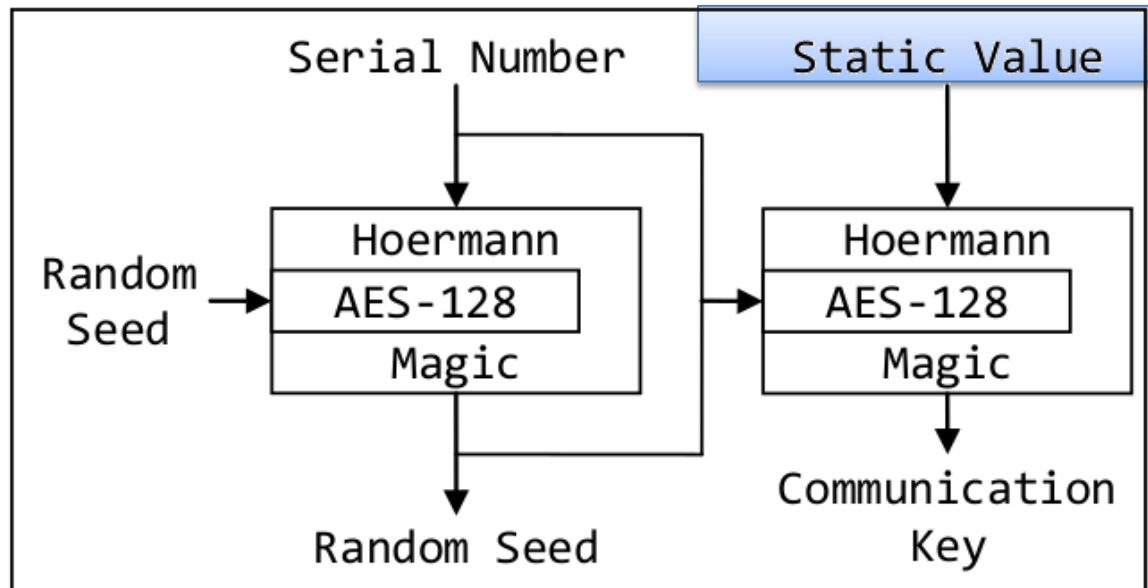
Field	Length (Byte)	Comment
Constant	1	0x70, 0x50
Serial Number	4	unique to device
Encrypted Data	16	adapted AES-128
Checksum	1	CRC



GENERATION OF COMMUNICATION KEYS

System Analysis

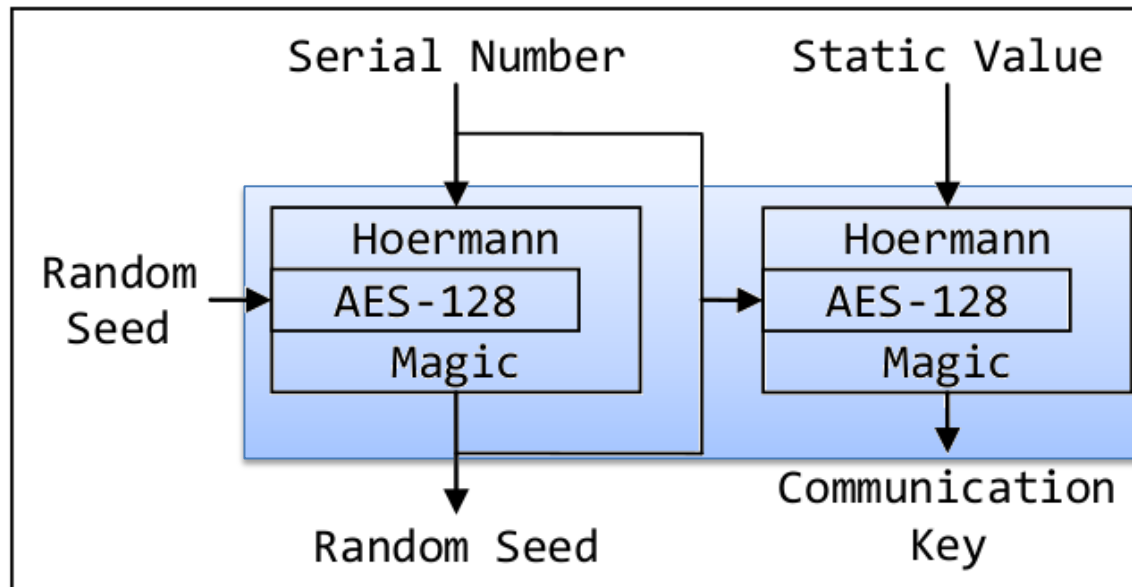
The static value is
hardcoded in the
firmware



System Analysis

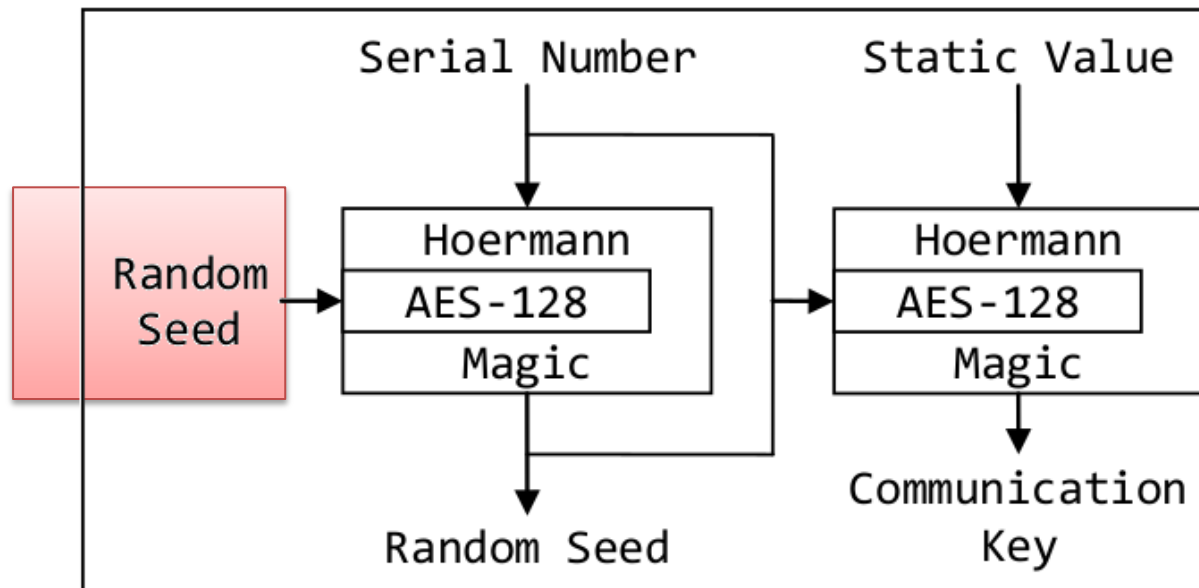
The encryption scheme is implemented in the firmware.

Kerckhoff's principle



System Analysis

Initial random seed
was identical on all our
devices

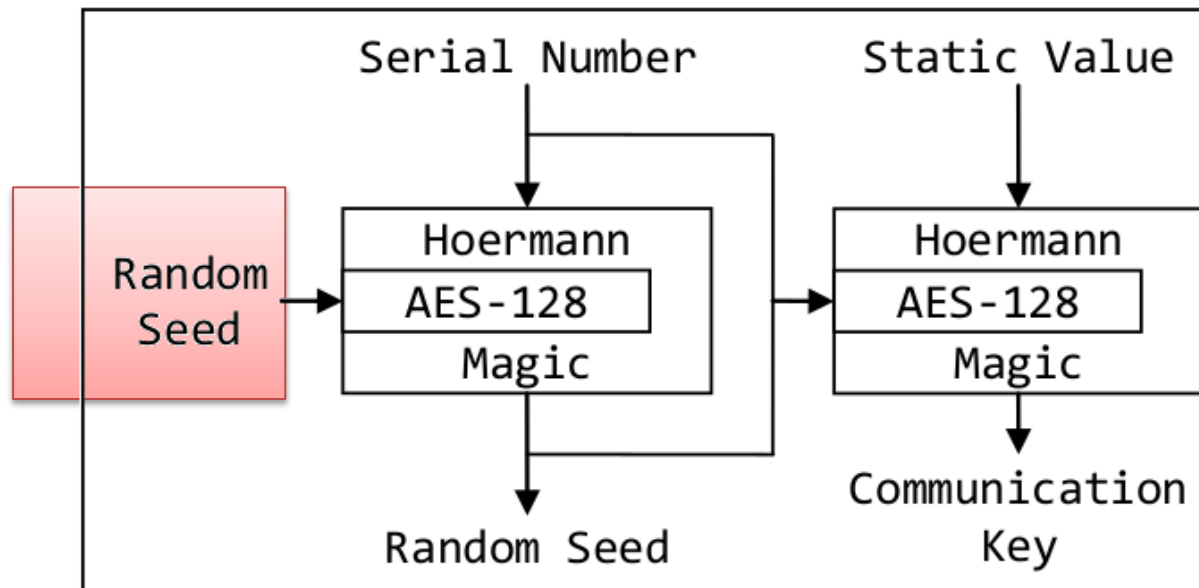


System Analysis

Initial random seed
was identical on all our
devices

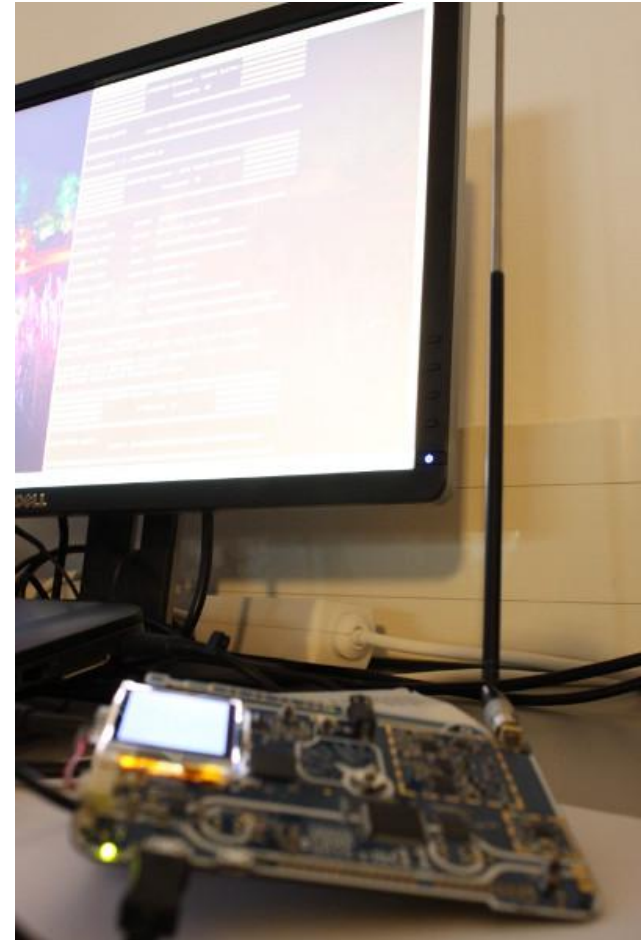


Vulnerability



How does the Attack Work ?

- Low-cost SDR such as the CCC rad1o or HackRF
- The CCC rad1o was available for free at the CCC Camp 2015 (“conference badge”)

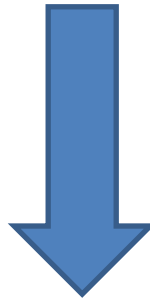


How does the Attack Work ?

- Step 1: Record the RF transmission from a BiSecur hand transmitter

How does the Attack Work ?

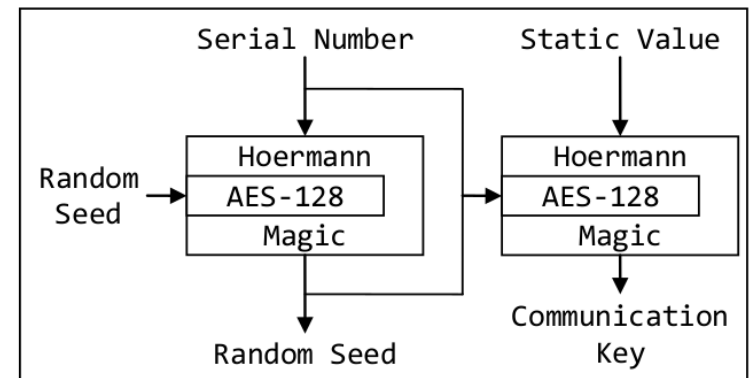
- Step 1: Record the RF transmission from a BiSecur hand transmitter



Field	Length (Byte)	Comment
Constant	1	0x70, 0x50
Serial Number	4	unique to device
Encrypted Data	16	adapted AES-128
Checksum	1	CRC

How does the Attack Work ?

- Step 2: Use the information obtained from an arbitrary hand transmitter* and the RF frame
- We know:
 - Encryption scheme (AES + "magic")
 - Static value
 - Initial random seed
 - Structure of a decrypted message
 - Serial number (from recorded RF frame)
 - Encrypted payload (from recorded RF frame)



GENERATION OF COMMUNICATION KEYS

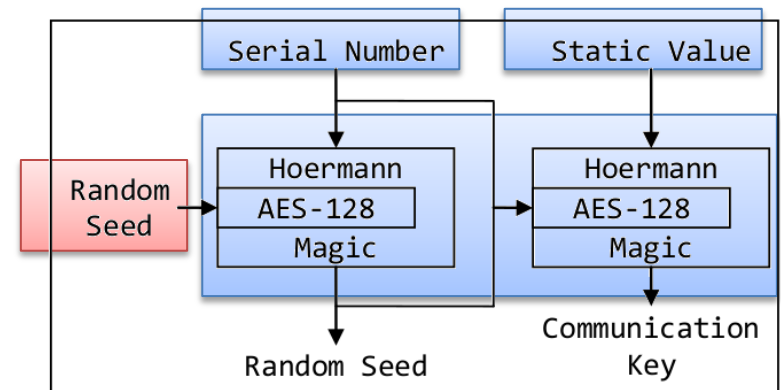
* to protect BiSecur customers, we do not disclose this information

How does the Attack Work ?

- Step 2: Use the information obtained from an arbitrary hand transmitter and the RF frame

- We know:

- Encryption scheme (AES + "magic")
- Static value
- Initial random seed
- Structure of a decrypted message
- Serial number (from recorded RF frame)
- Encrypted payload (from recorded RF frame)

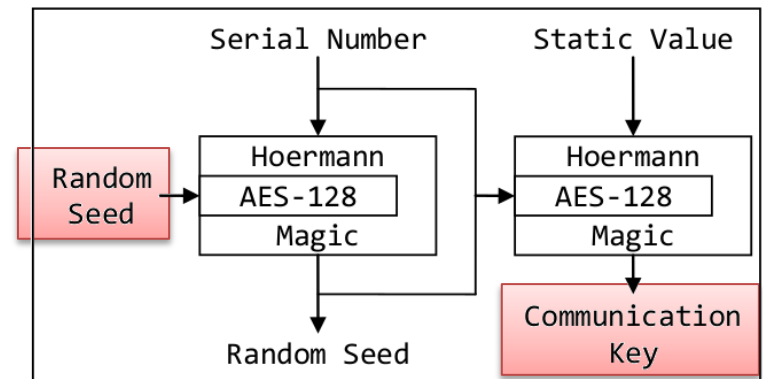


GENERATION OF COMMUNICATION KEYS

How does the Attack Work ?

- Step 3: Compute **Communication Key** candidate

- Candidate is correct if it has the expected plaintext structure

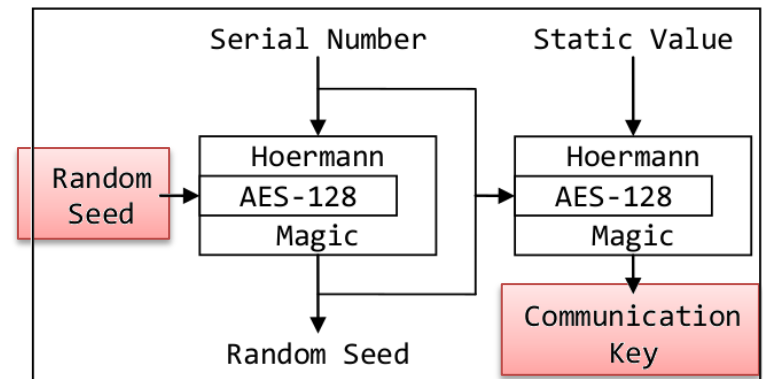


GENERATION OF COMMUNICATION KEYS

- Otherwise repeat (this is the case if the user has manually generated a new key)

How does the Attack Work ?

- Step 4: Obtain the current counter value from the decrypted message

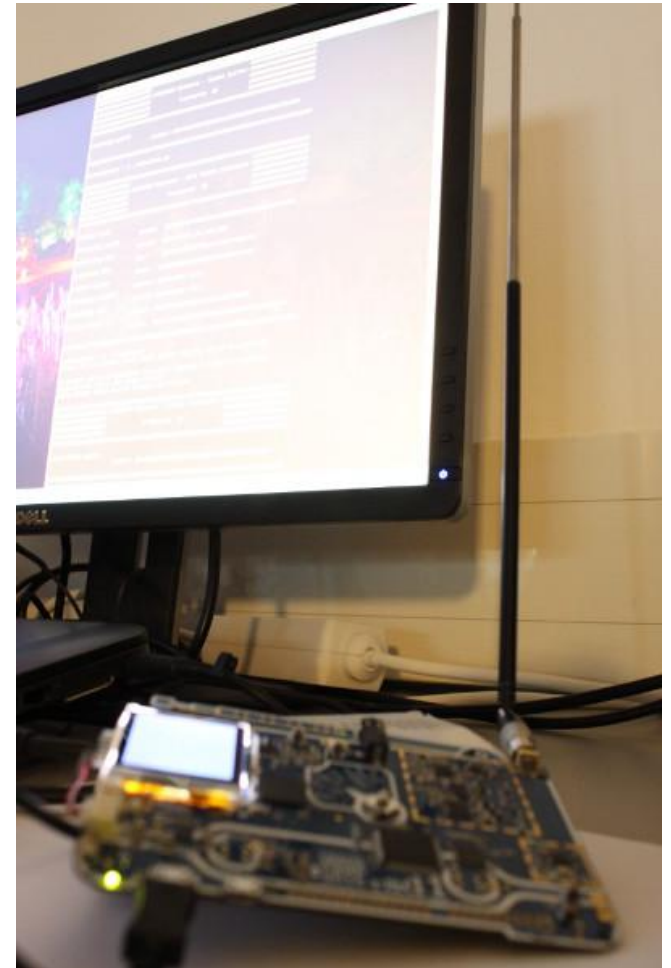


How does the Attack Work ?

- Step 5: Increase the counter value by one, encrypt the message with the obtained **Communication Key**

How does the Attack Work ?

- Step 6: Transmit RF frame, door should open
- Proof-of-Concept Demo



Impact Assessment

- Observation: serial numbers of same model hand transmitters bought at the same time were close to each other
- Assumption: Sequential serial numbers --> probably millions of devices in the field
- Not sure if our guess is correct !

0x043F3D68	71.253.352	HSE1 glossy (old working)
0x043F3D78	71.253.368	HSE1 glossy (old defect)
0x046972A0	74.019.488	HSE1 matte (new)
0x0404F1AB	67.432.875	HSE2 black (new)
0x046A2489	74.065.033	HSE2 white (new)
0x0462DD51	73,588,049	HSE2 in set (new)
0x0462DD5A	73.588.058	HSE2 in set (new)

How can the Vulnerability be Fixed ?

- Each hand transmitter needs to have its individual random seed value (80-bit)
- Since the random seed is no longer shared between all hand transmitters, an attacker can no longer compute the communication key without brute-forcing the 80-bit random seed

Responsible Disclosure



[<https://www.siteground.com/blog/responsible-disclosure>]

Responsible Disclosure

- 4.10.2017 - Involving the Austrian national CERT team as coordinator, we reported the security vulnerability including a detailed advisory and a suggested security fix so that the manufacturer can fix the issue
- 31.10.2017 – Confirmation from CERT that the manufacturer received and understood the security problem

Responsible Disclosure

- [...]: various e-mails and phone calls
- End of Nov. 2017: Meeting with manufacturer: we presented the vulnerability and the suggested security fix
- Dec. 2017: Security fix implemented and in testing phase

Conclusion

- We presented a viable methodology to analyze wireless RF systems with microcontrollers
- We believe that independent security audits are an essential tool to achieve a high level of security in a product
- When it comes to hardware security, it is good to have a Hardware Security lab at hand :-)
- We followed a responsible disclosure process and supported the manufacturer in understanding and fixing the vulnerability
- We will publish the security advisory after this talk (CVE ID: CVE-2017-17910, CVSSv2: 9.7)

Thank you for your Attention !

Trustworks KG
Rienoesslgasse 14/17
1040 Wien

Visit our homepage or contact us at:

Web: <https://www.trustworks.at>
E-Mail: office <at> trustworks.at