Hacking Toshiba Laptops

Or how to mess up your firmware security

•••

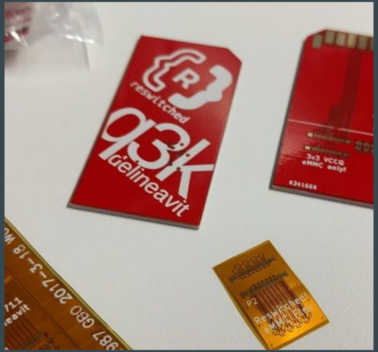
REcon Brussels 2018

whois

Serge Bazanski

Freelancer in devops & (hardware) security.

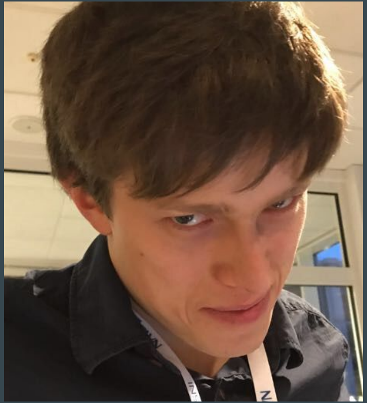
Twitter: @q3k
IRC: q3k @ freenode.net



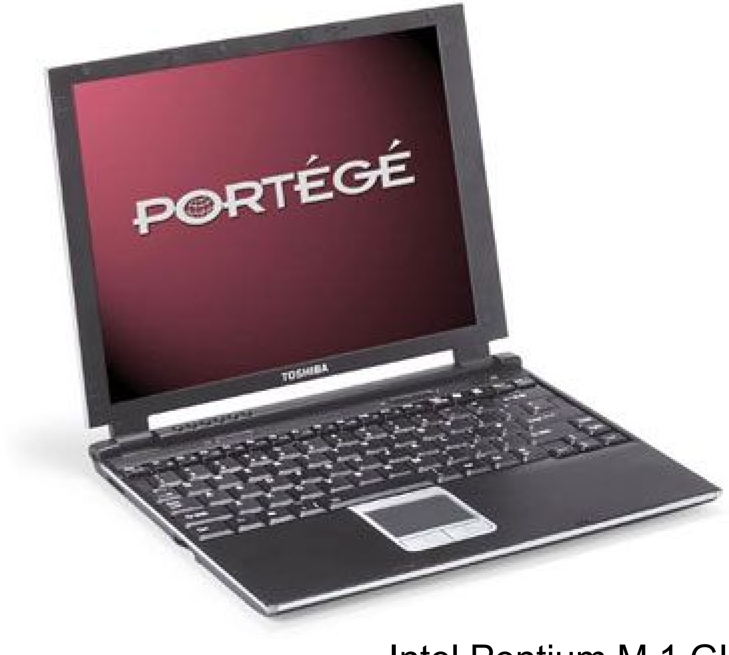
Michał Kowalczyk

Vice-captain @ Dragon Sector
Researcher @ Invisible Things Lab
Reverse engineer, amateur cryptanalyst

Twitter: @dsredford
IRC: Redford @ freenode.net

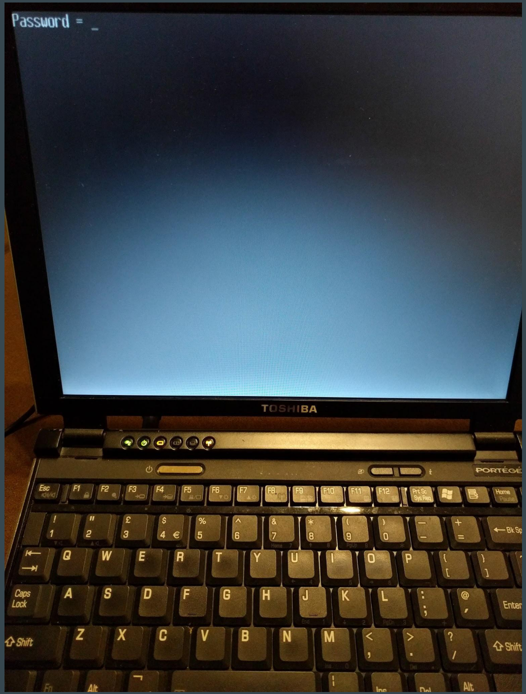


Toshiba Portégé R100

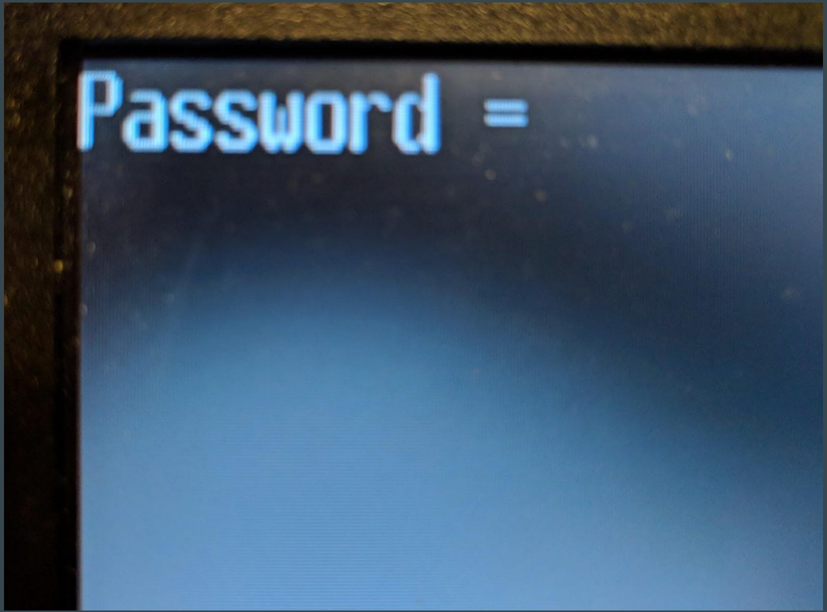


Intel Pentium M 1 GHz
256MB RAM

But there’s a catch…



```
Password = _
```



```
Password =
```

Quite the catch, actually.

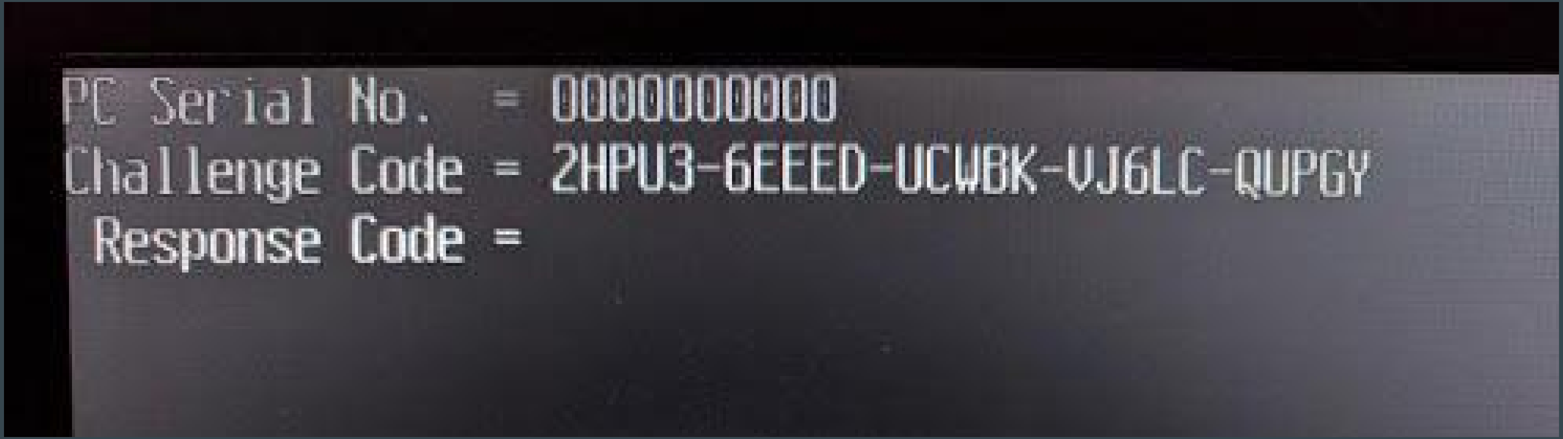
CMOS clear jumper? None to be found.

Yank out the battery? Password still there.

Take a door key and pass it over the pins of things that look like flash chips hopefully causing a checksum failure and resetting the password?

Nice try. No luck, though.

A-ha!



```
PC Serial No.  = 0000000000
Challenge Code = 2HPU3-6EEED-UCWBK-VJ6LC-QUPGY
 Response Code =
```

BIOS analysis

How to get the BIOS code?

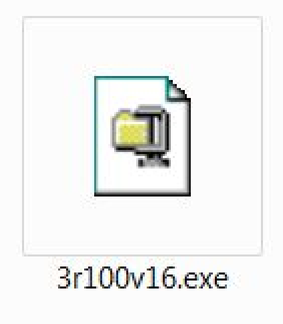
Physical memory? Not with a locked-down laptop.

Dump of the flash chip? Ugh.

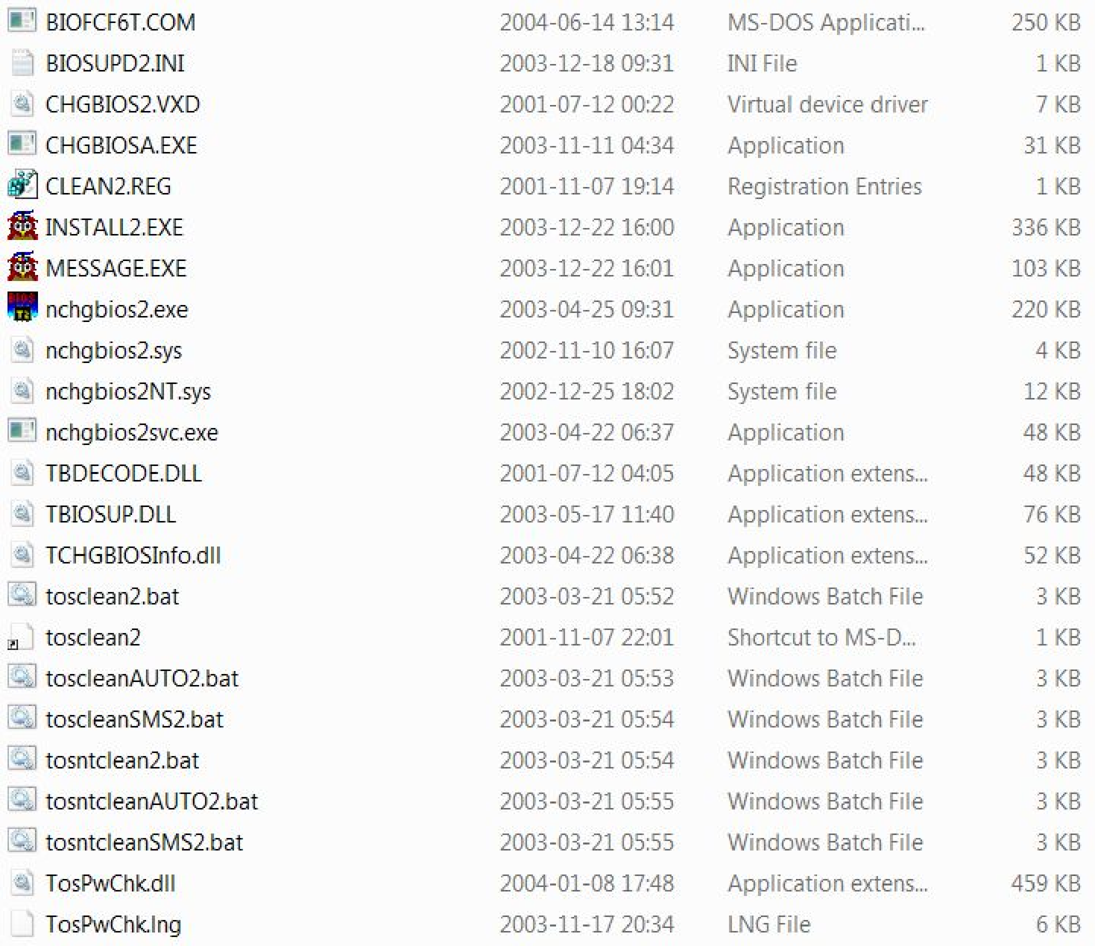
Unpack some updates? Let’s see.

Unpacking the updates

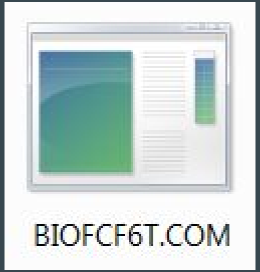
https://support.toshiba.com/ →



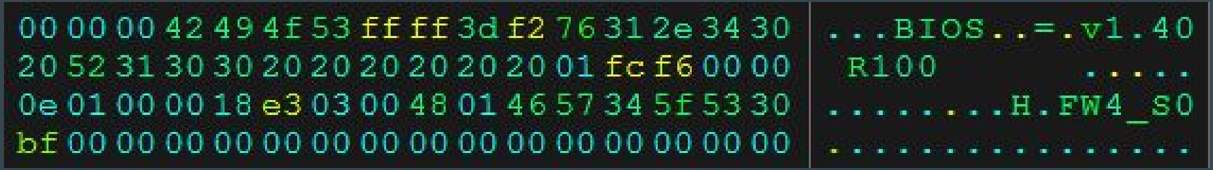
7-Zip



BIOFCF6T.COM	2004-06-14 13:14	MS-DOS Applicati...	250 KB
BIOSUPD2.INI	2003-12-18 09:31	INI File	1 KB
CHGBIOS2.VXD	2001-07-12 00:22	Virtual device driver	7 KB
CHGBIOSA.EXE	2003-11-11 04:34	Application	31 KB
CLEAN2.REG	2001-11-07 19:14	Registration Entries	1 KB
INSTALL2.EXE	2003-12-22 16:00	Application	336 KB
MESSAGE.EXE	2003-12-22 16:01	Application	103 KB
nchgbios2.exe	2003-04-25 09:31	Application	220 KB
nchgbios2.sys	2002-11-10 16:07	System file	4 KB
nchgbios2NT.sys	2002-12-25 18:02	System file	12 KB
nchgbios2svc.exe	2003-04-22 06:37	Application	48 KB
TBDECODE.DLL	2001-07-12 04:05	Application extens...	48 KB
TBIOSUP.DLL	2003-05-17 11:40	Application extens...	76 KB
TCHGBIOSInfo.dll	2003-04-22 06:38	Application extens...	52 KB
tosclean2.bat	2003-03-21 05:52	Windows Batch File	3 KB
tosclean2	2001-11-07 22:01	Shortcut to MS-D...	1 KB
toscleanAUTO2.bat	2003-03-21 05:53	Windows Batch File	3 KB
toscleanSMS2.bat	2003-03-21 05:54	Windows Batch File	3 KB
tosntclean2.bat	2003-03-21 05:54	Windows Batch File	3 KB
tosntcleanAUTO2.bat	2003-03-21 05:55	Windows Batch File	3 KB
tosntcleanSMS2.bat	2003-03-21 05:55	Windows Batch File	3 KB
TosPwChk.dll	2004-01-08 17:48	Application extens...	459 KB
TosPwChk.lng	2003-11-17 20:34	LNG File	6 KB



BIOFCF6T.COM



```
00 00 00 42 49 4f 53 ff ff 3d f2 76 31 2e 34 30  ...BIOS..=.v1.40
20 52 31 30 30 20 20 20 20 20 20 01 fc f6 00 00   R100      .....
0e 01 00 00 18 e3 03 00 48 01 46 57 34 5f 53 30  ........H.FW4_S0
bf 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

+

254 KB of compressed data

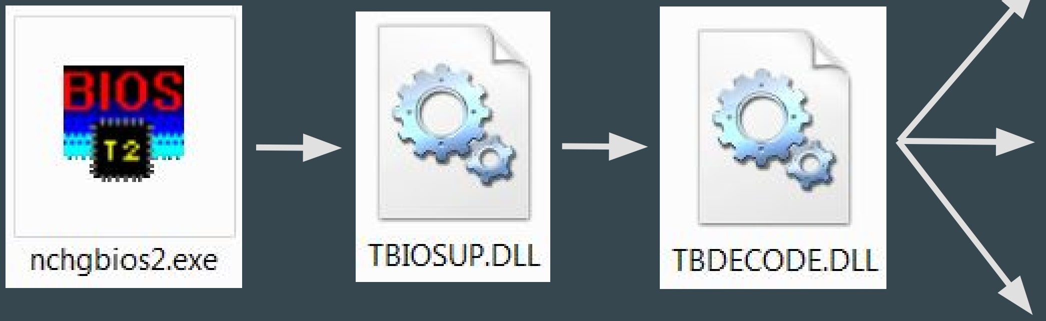
Decompression

Unknown format

Default unpacker is a 16-bit EXE

There’s an alternative one, 32-bit!

Decompression



BuIsFileCompressed

BuGetFileSize

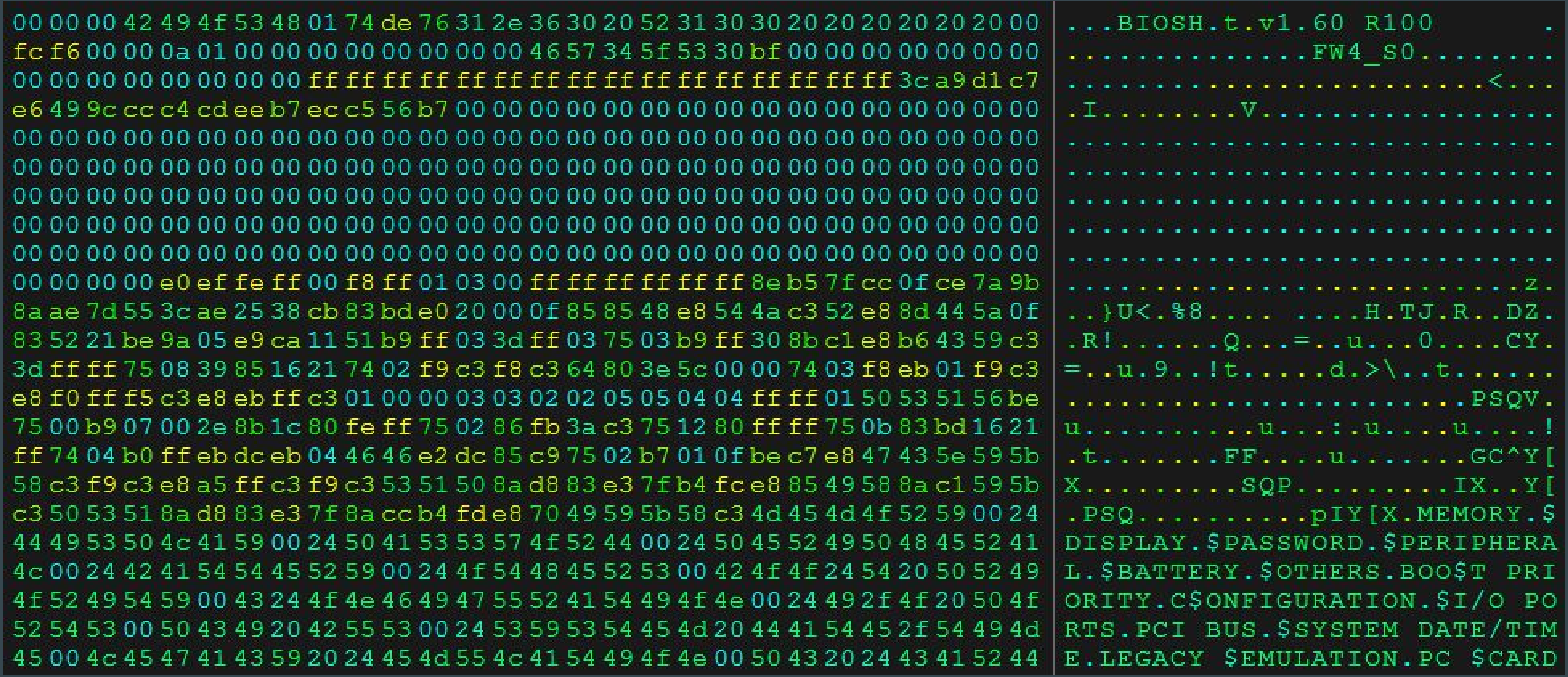
BuDecodeFile

Decompression

Just ~50 lines of C!

```c
...
BuIsFileCompressed(compressed, &is_compressed);
if (is_compressed) {
   BuDecodeFile(compressed, fsize, decompressed);
}
...
```

The result

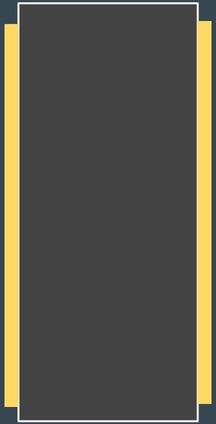


```
00 00 00 42 49 4f 53 48 01 74 de 76 31 2e 36 30 20 52 31 30 30 20 20 20 20 20 20 00 | ...BIOSH.t.v1.60 R100      .
fc f6 00 00 0a 01 00 00 00 00 00 00 00 00 46 57 34 5f 53 30 bf 00 00 00 00 00 00 00 | ..............FW4_S0........
00 00 00 00 00 00 00 00 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff 3c a9 d1 c7 | ........................<...
e6 49 9c cc c4 cd ee b7 ec c5 56 b7 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .I........V.................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ............................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ............................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ............................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ............................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ............................
00 00 00 00 e0 ef fe ff 00 f8 ff 01 03 00 ff ff ff ff ff ff 8e b5 7f cc 0f ce 7a 9b | ..........................z.
8a ae 7d 55 3c ae 25 38 cb 83 bd e0 20 00 0f 85 85 48 e8 54 4a c3 52 e8 8d 44 5a 0f | ..}U<.%8.... ....H.TJ.R..DZ.
83 52 21 be 9a 05 e9 ca 11 51 b9 ff 03 3d ff 03 75 03 b9 ff 30 8b c1 e8 b6 43 59 c3 | .R!......Q...=..u...0....CY.
3d ff ff 75 08 39 85 16 21 74 02 f9 c3 f8 c3 64 80 3e 5c 00 00 74 03 f8 eb 01 f9 c3 | =..u.9..!t.....d.>\..t......
e8 f0 ff f5 c3 e8 eb ff c3 01 00 00 03 03 02 02 05 05 04 04 ff ff 01 50 53 51 56 be | .......................PSQV.
75 00 b9 07 00 2e 8b 1c 80 fe ff 75 02 86 fb 3a c3 75 12 80 ff ff 75 0b 83 bd 16 21 | u..........u...:.u....u....!
ff 74 04 b0 ff eb dc eb 04 46 46 e2 dc 85 c9 75 02 b7 01 0f be c7 e8 47 43 5e 59 5b | .t.......FF....u.......GC^Y[
58 c3 f9 c3 e8 a5 ff c3 f9 c3 53 51 50 8a d8 83 e3 7f b4 fc e8 85 49 58 8a c1 59 5b | X.........SQP.........IX..Y[
c3 50 53 51 8a d8 83 e3 7f 8a cc b4 fd e8 70 49 59 5b 58 c3 4d 45 4d 4f 52 59 00 24 | .PSQ..........pIY[X.MEMORY.$
44 49 53 50 4c 41 59 00 24 50 41 53 53 57 4f 52 44 00 24 50 45 52 49 50 48 45 52 41 | DISPLAY.$PASSWORD.$PERIPHERA
4c 00 24 42 41 54 54 45 52 59 00 24 4f 54 48 45 52 53 00 42 4f 4f 24 54 20 50 52 49 | L.$BATTERY.$OTHERS.BOO$T PRI
4f 52 49 54 59 00 43 24 4f 4e 46 49 47 55 52 41 54 49 4f 4e 00 24 49 2f 4f 20 50 4f | ORITY.C$ONFIGURATION.$I/O PO
52 54 53 00 50 43 49 20 42 55 53 00 24 53 59 53 54 45 4d 20 44 41 54 45 2f 54 49 4d | RTS.PCI BUS.$SYSTEM DATE/TIM
45 00 4c 45 47 41 43 59 20 24 45 4d 55 4c 41 54 49 4f 4e 00 50 43 20 24 43 41 52 44 | E.LEGACY $EMULATION.PC $CARD
```

Dumping the BIOS flash

Where to start looking

Chip Safari



RAM

Flash

Google it

Interfacing to flash chips

In-circuit: test pads or protocol that permits multi-master access

Out-of-circuit (?): desolder, attach to breakout/clip, use main communication interface

Custom breakout board

KiCAD (or $whatever, really) PCB design.

Thermal transfer for DIY PCB manufacturing.

Hot air gun to desolder, soldering station to re-solder.

Tools you’ll need

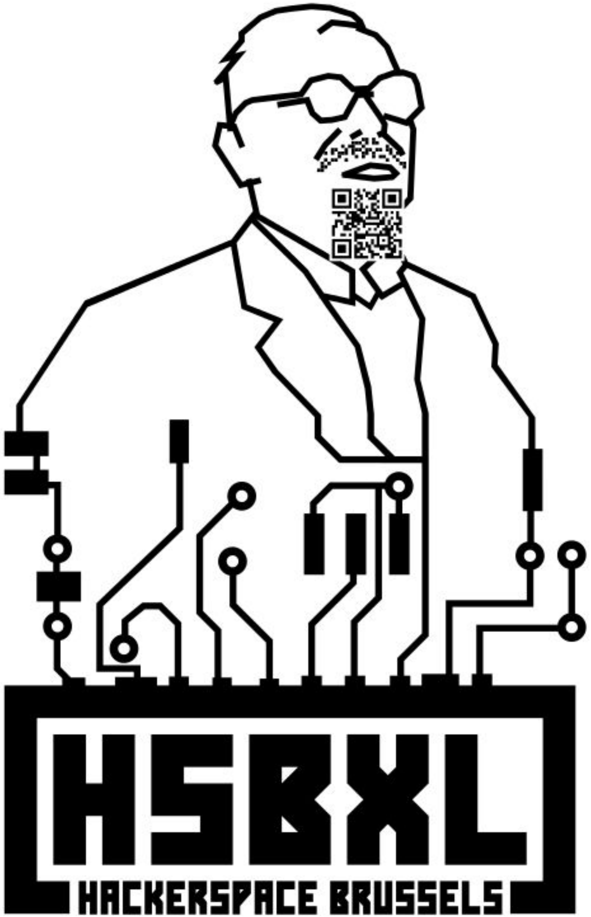
3eur

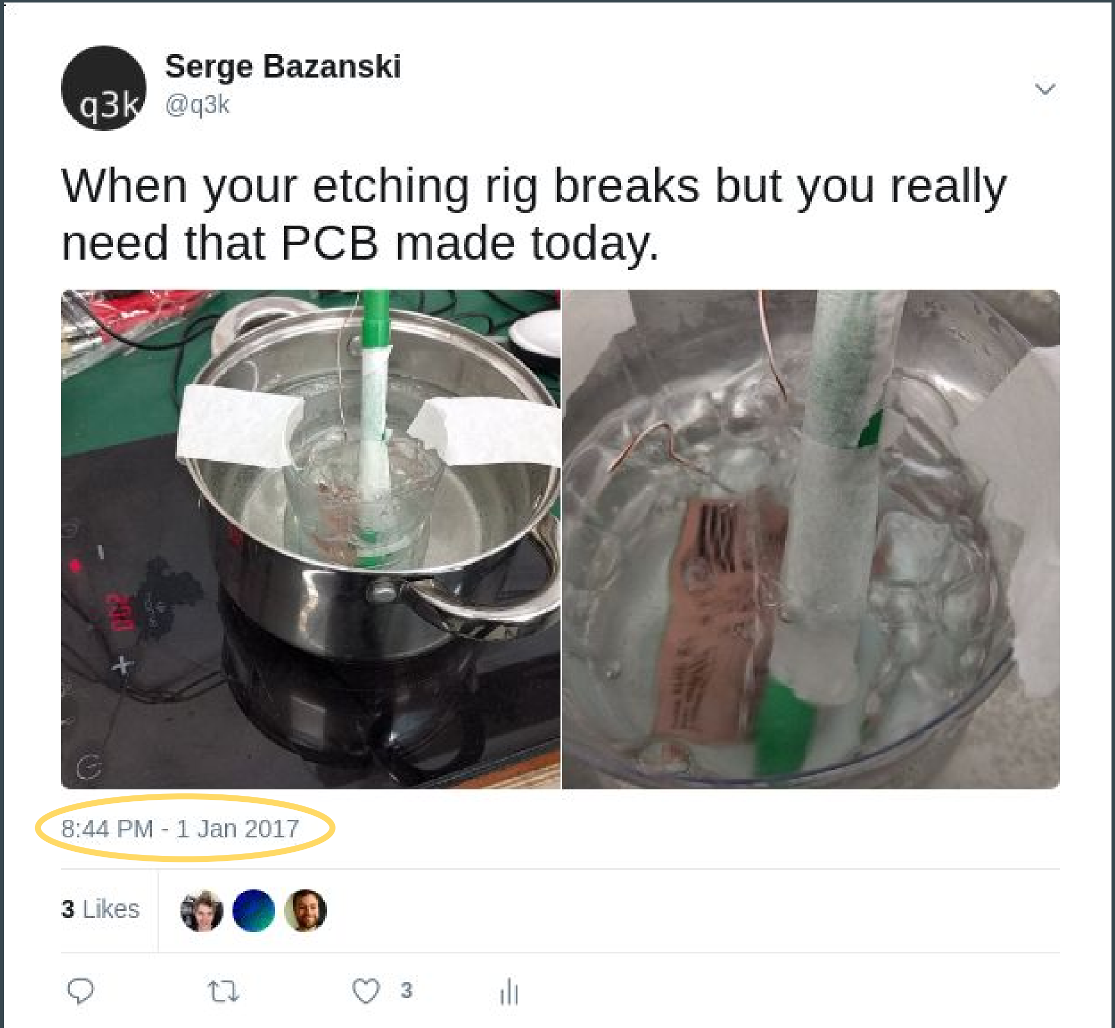
150eur

50eur

Hackerspace

25eur p/m + BYOB





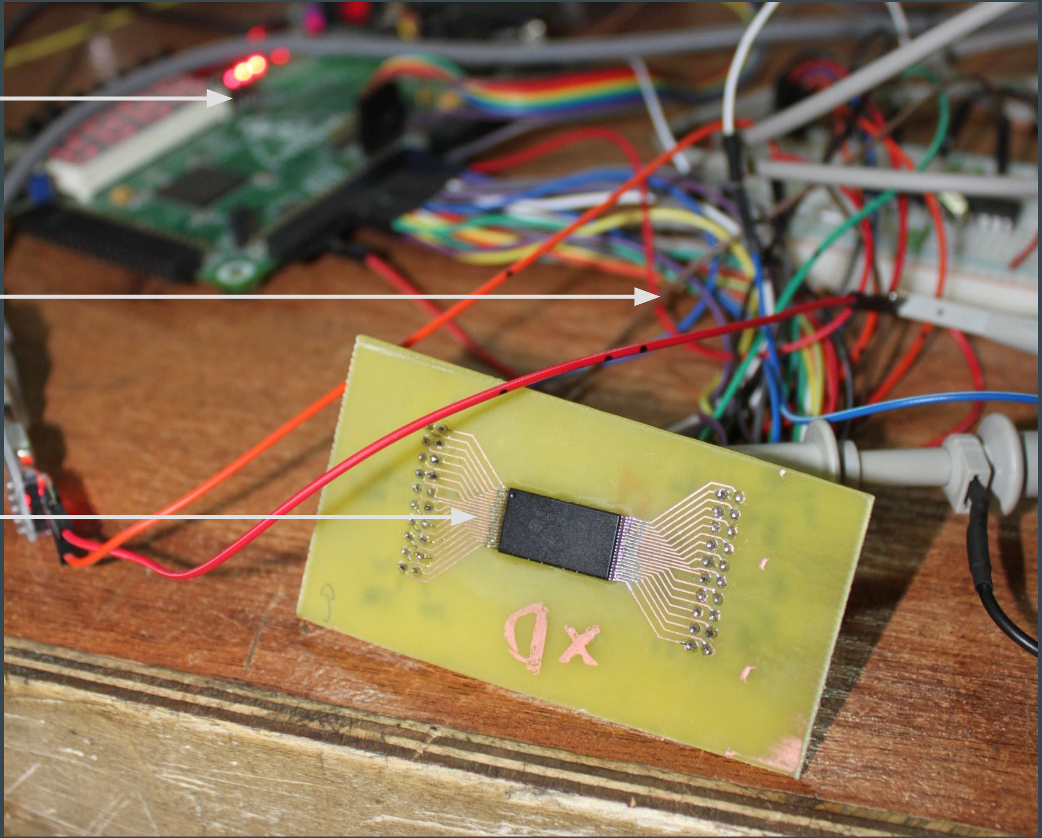
Serge Bazanski
@q3k

When your etching rig breaks but you really need that PCB made today.

8:44 PM - 1 Jan 2017

3 Likes

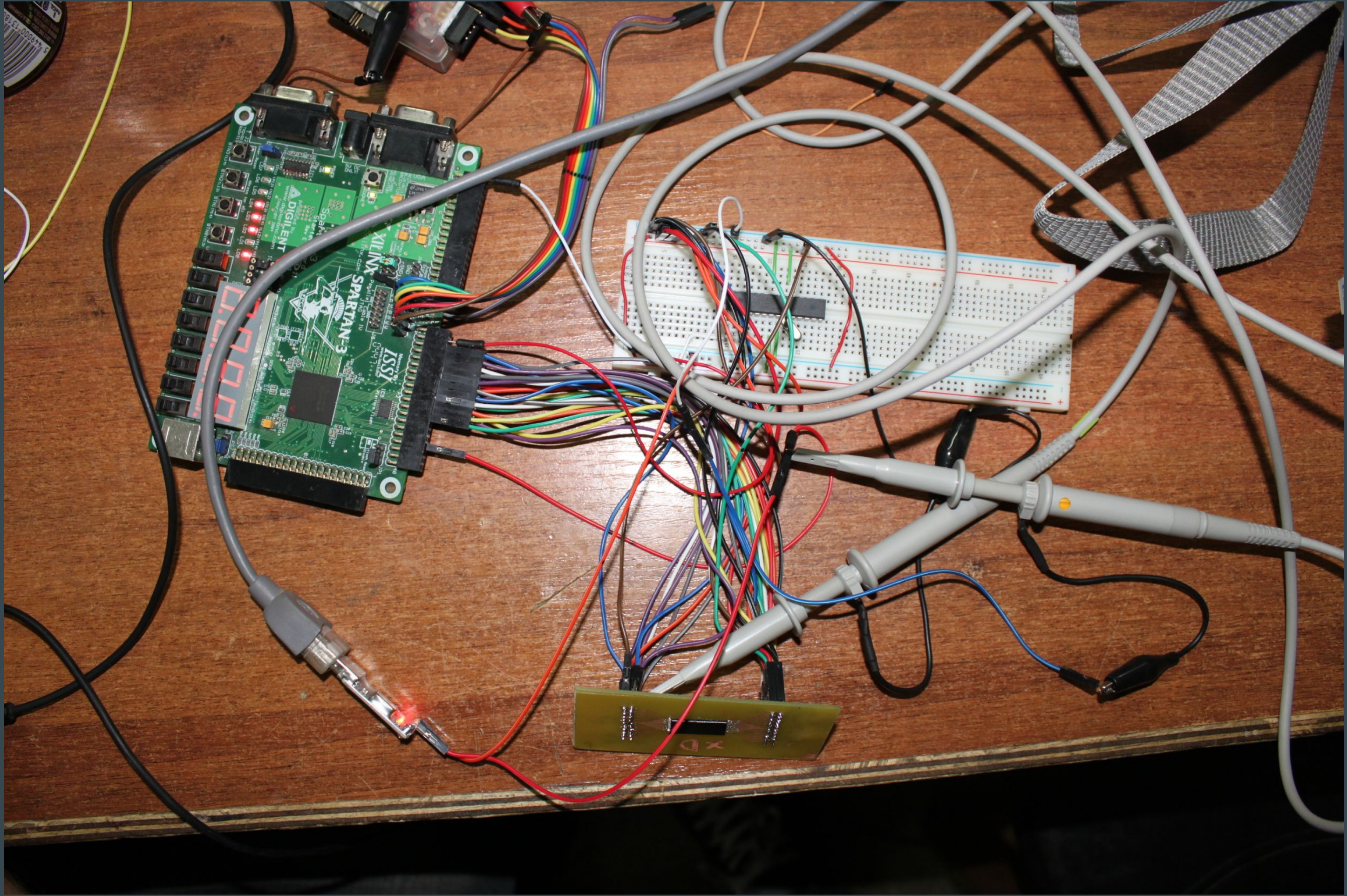
3



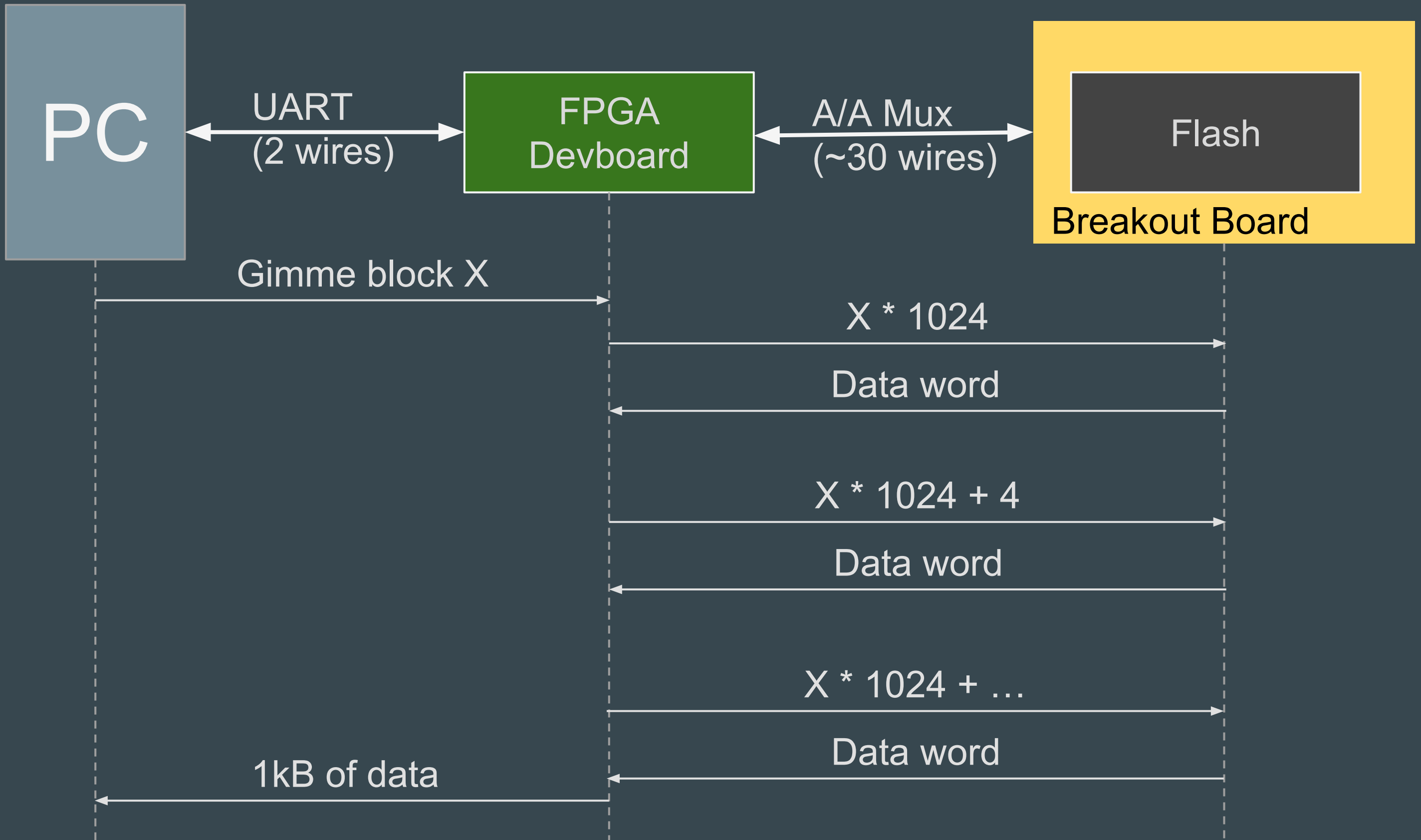
FPGA board
(Spartan 3E)

Kabelsalat

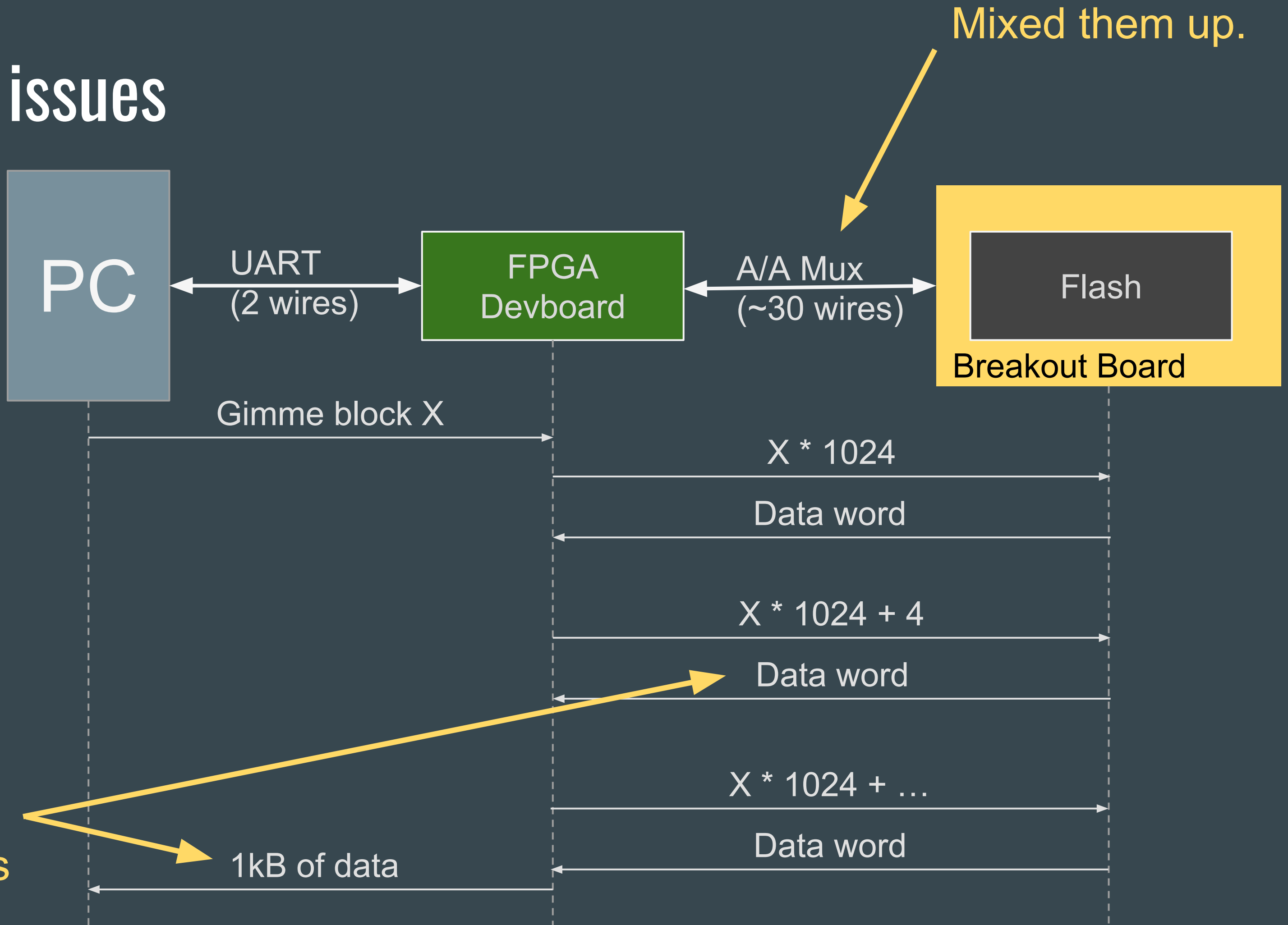
Flash


DIGILENT
XILINX
SPARTAN-3
ISSI

Setup


PC
UART
(2 wires)
FPGA
Devboard
A/A Mux
(~30 wires)
Flash
Breakout Board
Gimme block X
X * 1024
Data word
X * 1024 + 4
Data word
X * 1024 + …
Data word
1kB of data

Setup issues



Forgot checksums

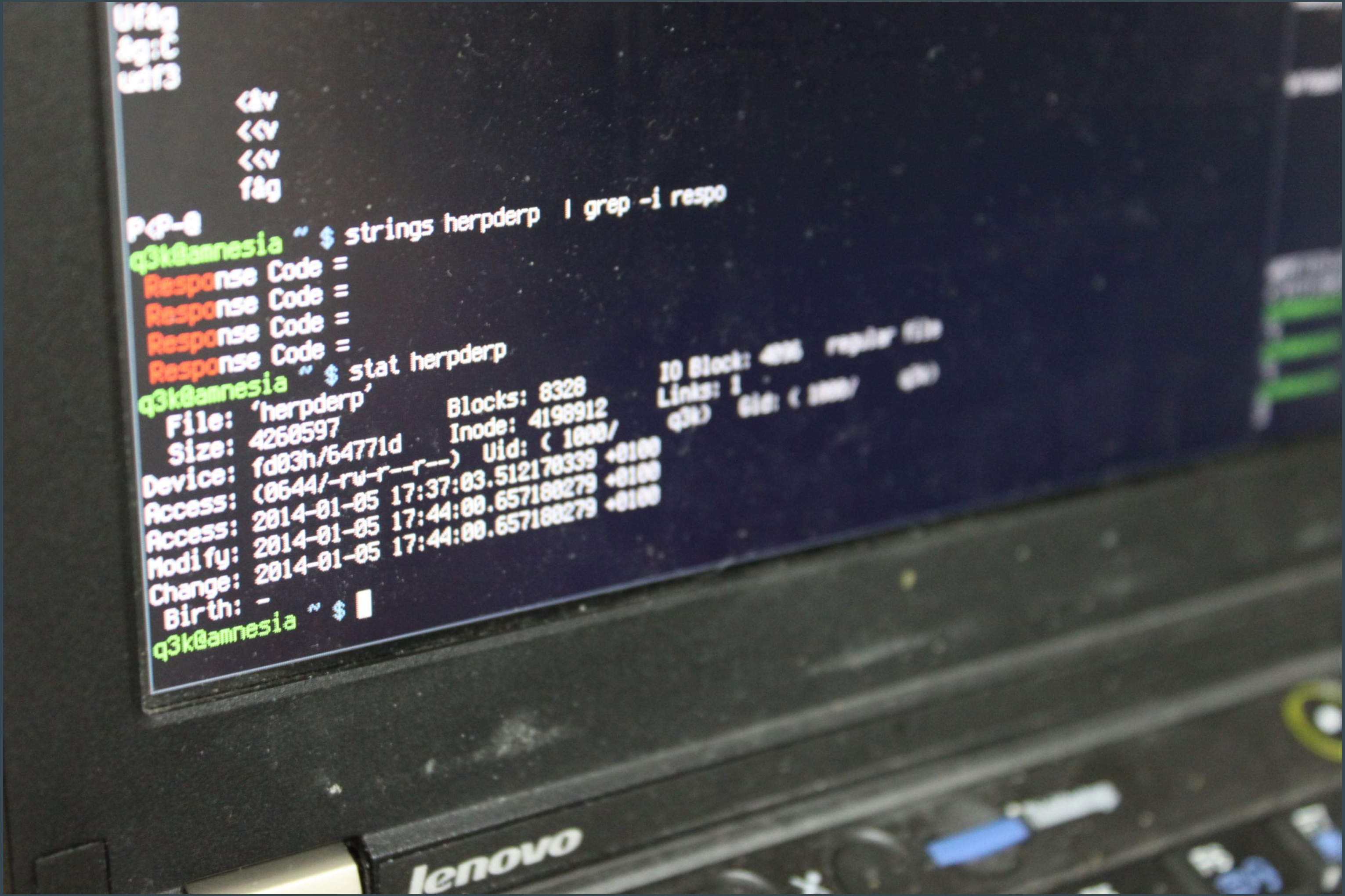
But why the FPGA?

Using an FPGA was unnecessary - just needed a bunch of I/O.

Comparatively difficult to develop for. And to debug.

Should’ve gone for a uC with a bunch of I/O or with a multiplexer.

But at least now we know ¯_(ツ)_/¯ .



```
Uf&g
&g:C
udf3
        <&v
        <<v
        <<v
        f&g
P<P-@
q3k@amnesia ~ $ strings herpderp  | grep -i respo
 Response Code =
 Response Code =
 Response Code =
 Response Code =
q3k@amnesia ~ $ stat herpderp
  File: 'herpderp'
  Size: 4260597         Blocks: 8328       IO Block: 4096   regular file
Device: fd03h/64771d    Inode: 4198912     Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1000/     q3k)   Gid: ( 1000/     q3k)
Access: 2014-01-05 17:37:03.512170339 +0100
Access: 2014-01-05 17:44:00.657180279 +0100
Modify: 2014-01-05 17:44:00.657180279 +0100
Change: 2014-01-05
 Birth: -
q3k@amnesia ~ $ 
```

BIOS code analysis

How to start?

CPU mode?

Entry point?

Memory map?

CPU start

> *“A hardware reset sets each processor’s registers to a known state and places the processor in real-address mode.”*

Intel® 64 and IA-32 Architectures
Software Developer’s Manual Volume 3

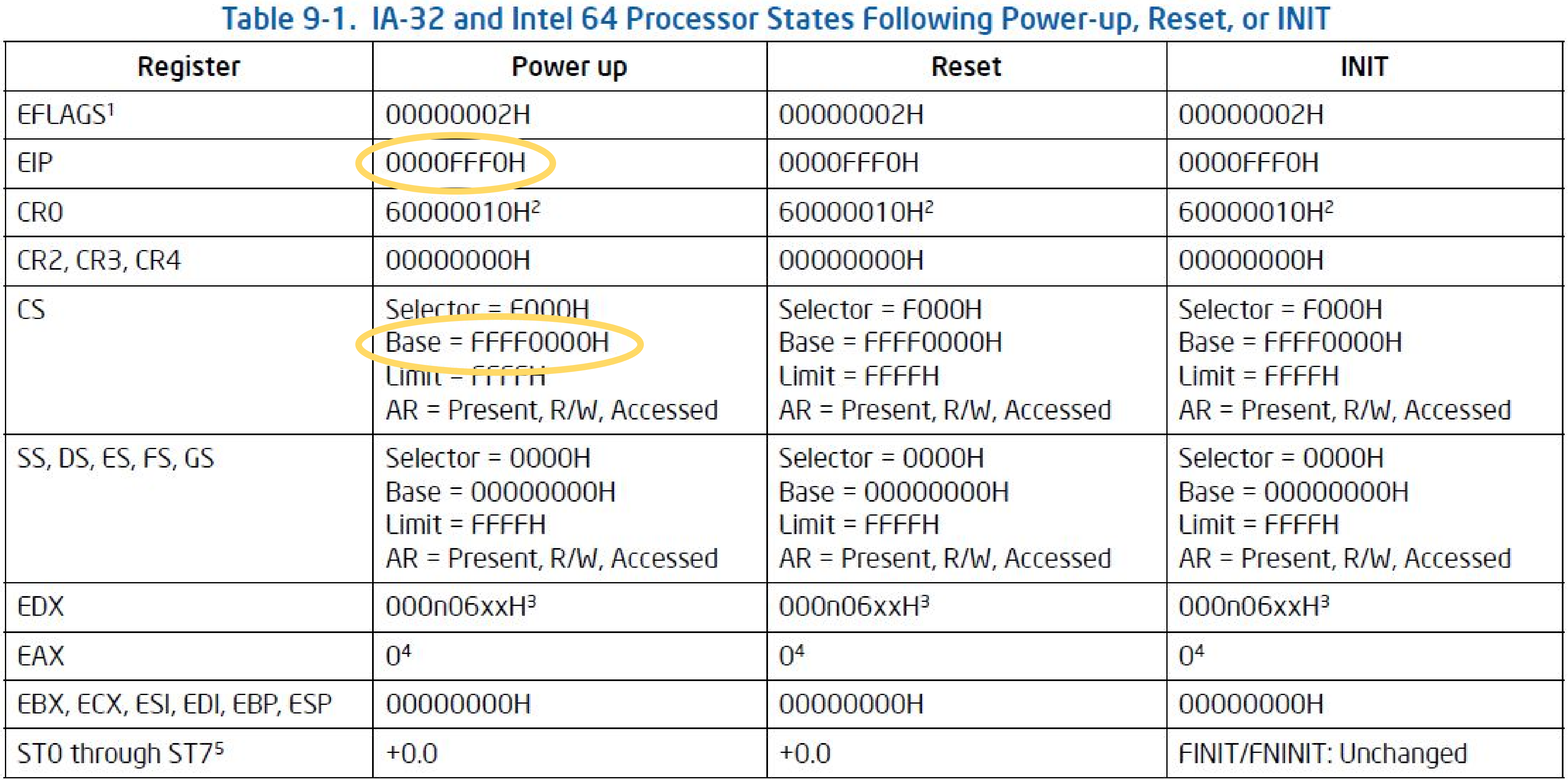


Table 9-1. IA-32 and Intel 64 Processor States Following Power-up, Reset, or INIT

Register	Power up	Reset	INIT
EFLAGS[1]	00000002H	00000002H	00000002H
EIP	0000FFF0H	0000FFF0H	0000FFF0H
CR0	60000010H[2]	60000010H[2]	60000010H[2]
CR2, CR3, CR4	00000000H	00000000H	00000000H

CS	Selector = F000H
	Base = FFFF0000H
	Limit = FFFFH
	AR = Present, R/W, Accessed
		Selector = F000H
		Base = FFFF0000H
		Limit = FFFFH
		AR = Present, R/W, Accessed
			Selector = F000H
			Base = FFFF0000H
			Limit = FFFFH
			AR = Present, R/W, Accessed

SS, DS, ES, FS, GS	Selector = 0000H
	Base = 00000000H
	Limit = FFFFH
	AR = Present, R/W, Accessed
		Selector = 0000H
		Base = 00000000H
		Limit = FFFFH
		AR = Present, R/W, Accessed
			Selector = 0000H
			Base = 00000000H
			Limit = FFFFH
			AR = Present, R/W, Accessed

EDX	000n06xxH[3]	000n06xxH[3]	000n06xxH[3]
EAX	0[4]	0[4]	0[4]
EBX, ECX, ESI, EDI, EBP, ESP	00000000H	00000000H	00000000H
ST0 through ST7[5]	+0.0	+0.0	FINIT/FNINIT: Unchanged

CPU start

We start at the address:

`CS:EIP = CS.Base + EIP = 0xFFFFFFF0`

Real Mode ⇒ physical address. A20 enabled.

So, what’s there?

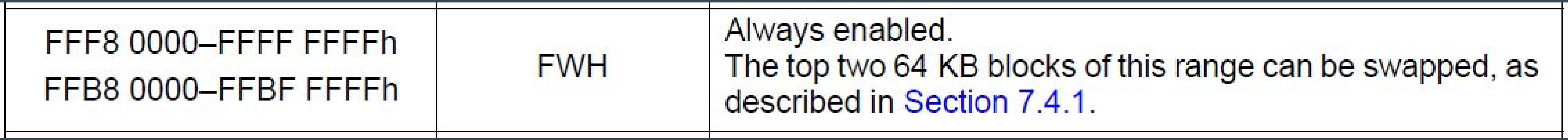
Memory mapping

Northbridge: Intel Odem MCH-M

No info about that region ⇒ let’s check the southbridge

Memory mapping

Southbridge: Intel ICH4-M

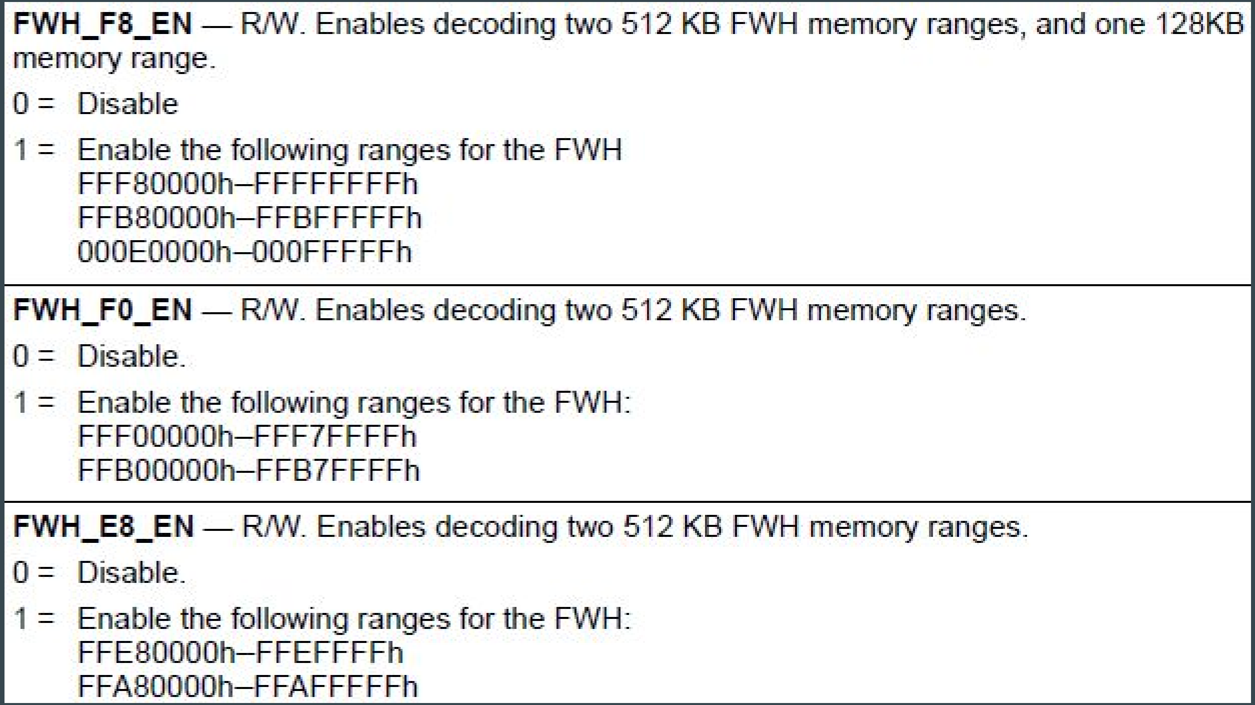


FFF8 0000–FFFF FFFFh FFB8 0000–FFBF FFFFh	FWH	Always enabled. The top two 64 KB blocks of this range can be swapped, as described in Section 7.4.1.

FWH = Firmware Hub = BIOS flash

Out dump has exactly 0x80000 bytes!

Even more mappings...



FWH_F8_EN — R/W. Enables decoding two 512 KB FWH memory ranges, and one 128KB memory range.

0 = Disable

1 = Enable the following ranges for the FWH
FFF80000h–FFFFFFFFh
FFB80000h–FFBFFFFFh
000E0000h–000FFFFFh

FWH_F0_EN — R/W. Enables decoding two 512 KB FWH memory ranges.

0 = Disable.

1 = Enable the following ranges for the FWH:
FFF00000h–FFF7FFFFh
FFB00000h–FFB7FFFFh

FWH_E8_EN — R/W. Enables decoding two 512 KB FWH memory ranges.

0 = Disable.

1 = Enable the following ranges for the FWH:
FFE80000h–FFEFFFFFh
FFA80000h–FFAFFFFFh

• • •

Entry point

```
FFFFFFF0:   jmp far FC00:3FA0

000FFFA0:   jmp far FC00:00A2

000FC0A2:   cli
000FC0A3:   cld
000FC0A4:   mov al, 2
000FC0A6:   out 92h, al ; Enable A20
                        ...
```

BIOS RE: Initialization

No stack! (and also no RAM)

16-bit Protected Mode + Unreal Mode

Checksums

RAM initialization

Self-copying into RAM

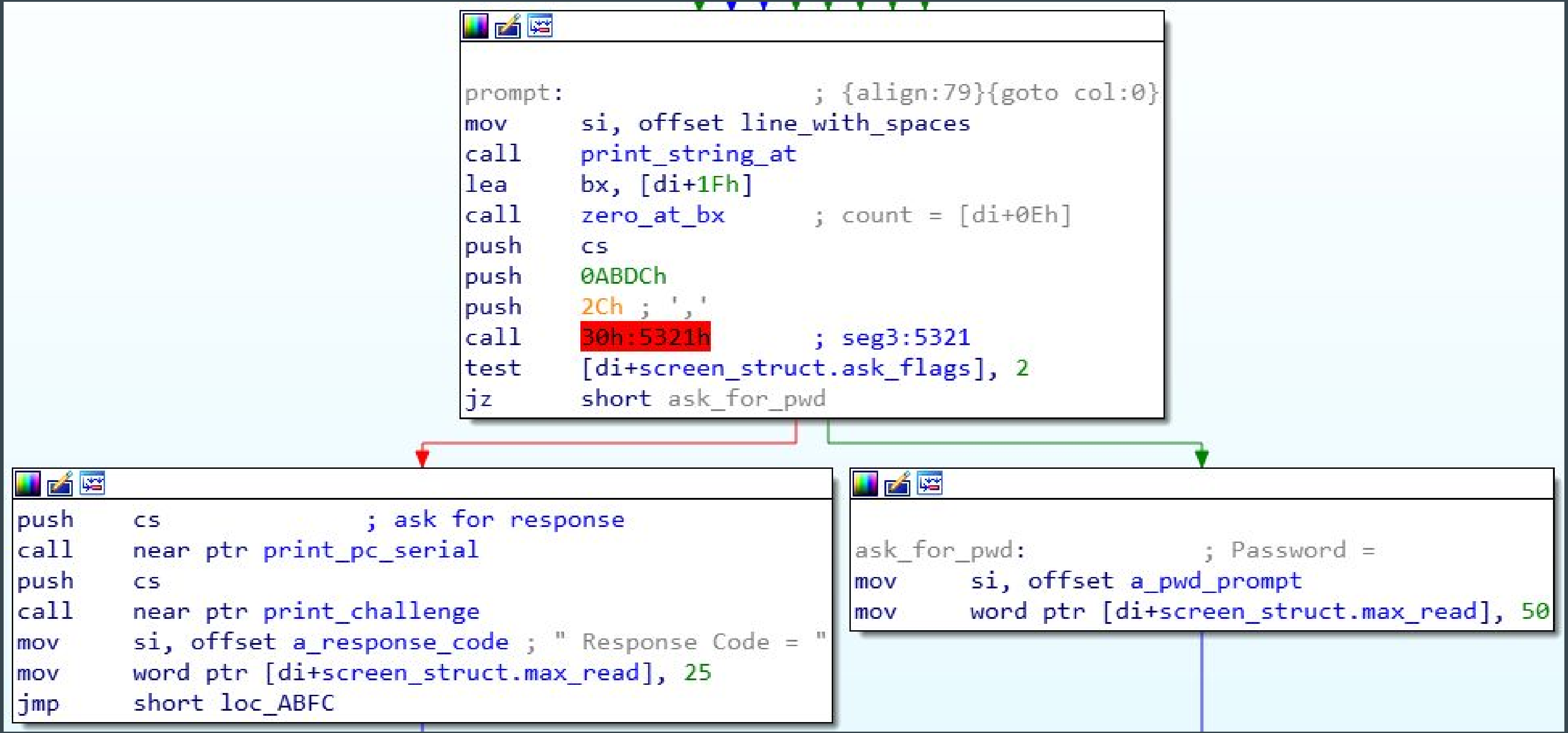
BIOS RE: Initialization

16-bit Protected Mode → segments!

We have to find and parse GDT

Only then we can analyze the code

BIOS RE: The password check



```
prompt:                         ; {align:79}{goto col:0}
mov     si, offset line_with_spaces
call    print_string_at
lea     bx, [di+1Fh]
call    zero_at_bx              ; count = [di+0Eh]
push    cs
push    0ABDCh
push    2Ch ; ','
call    30h:5321h               ; seg3:5321
test    [di+screen_struct.ask_flags], 2
jz      short ask_for_pwd
```

```
push    cs                      ; ask for response
call    near ptr print_pc_serial
push    cs
call    near ptr print_challenge
mov     si, offset a_response_code ; " Response Code = "
mov     word ptr [di+screen_struct.max_read], 25
jmp     short loc_ABFC
```

```
ask_for_pwd:                    ; Password =
mov     si, offset a_pwd_prompt
mov     word ptr [di+screen_struct.max_read], 50
```

BIOS RE: The password check

Everything eventually lands up in one function

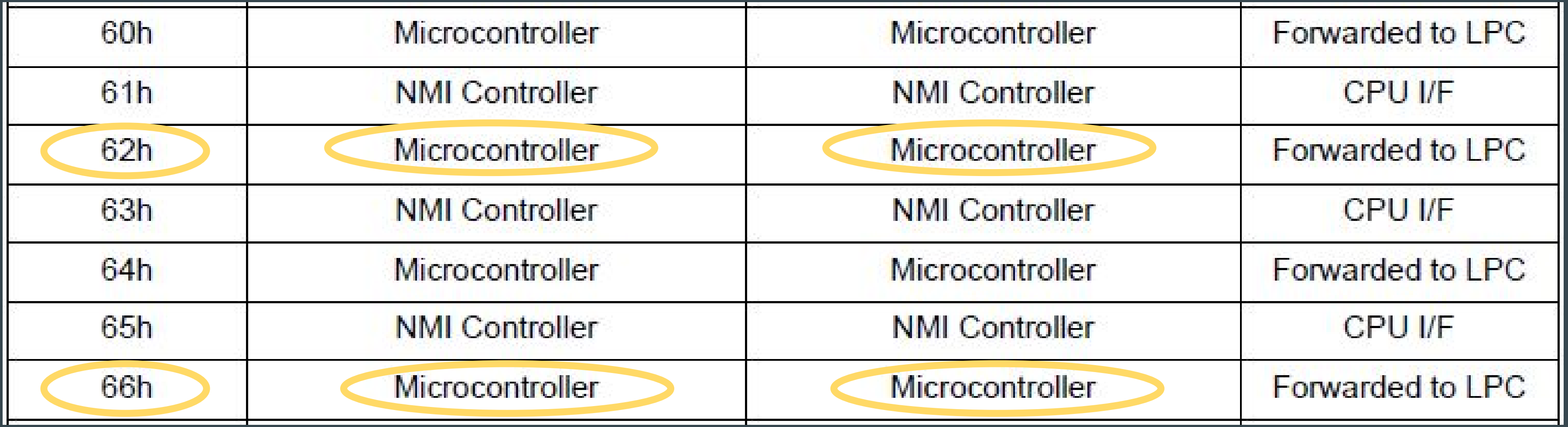
`f(in_buf) → out_buf`

After long analysis: all bytes are sent to I/O ports

62h and 66h

BIOS RE: The password check

From the southbridge manual:

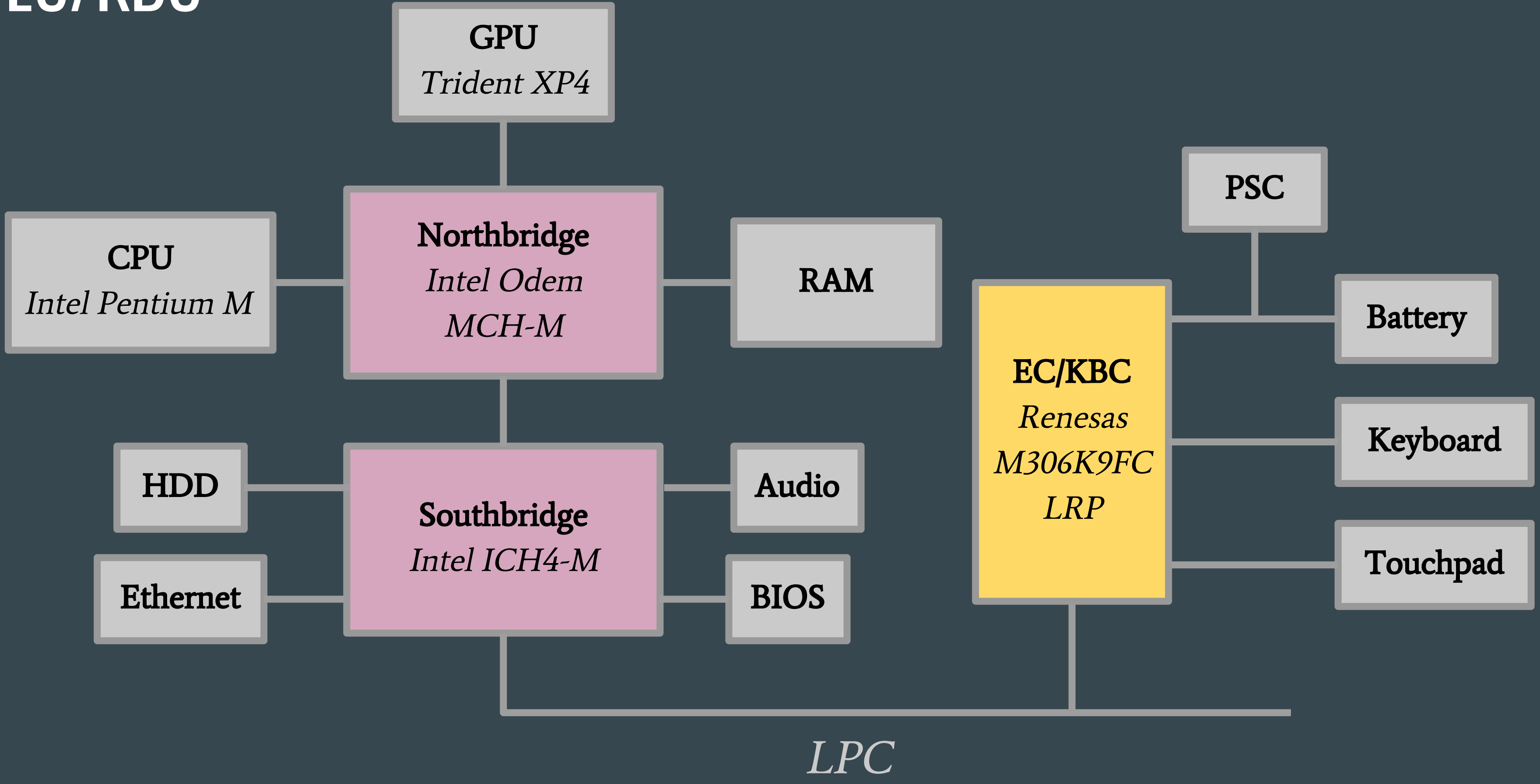


60h	Microcontroller	Microcontroller	Forwarded to LPC
61h	NMI Controller	NMI Controller	CPU I/F
62h	Microcontroller	Microcontroller	Forwarded to LPC
63h	NMI Controller	NMI Controller	CPU I/F
64h	Microcontroller	Microcontroller	Forwarded to LPC
65h	NMI Controller	NMI Controller	CPU I/F
66h	Microcontroller	Microcontroller	Forwarded to LPC

Table 6-2. Fixed I/O Ranges Decoded by Intel ICH4

“Microcontroller”???

EC/KBC


GPU
Trident XP4
CPU
Intel Pentium M
Northbridge
Intel Odem MCH-M
RAM
HDD
Ethernet
Southbridge
Intel ICH4-M
Audio
BIOS
PSC
Battery
EC/KBC
Renesas M306K9FC LRP
Keyboard
Touchpad
LPC

EC: Dump

How to obtain the code?

Updates!

EC: Dump

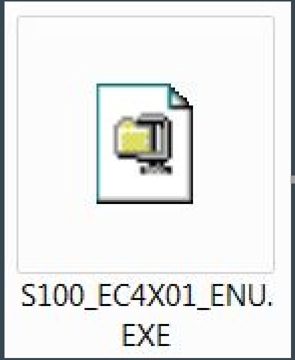
No updates available

BIOS changelog: nothing about the EC

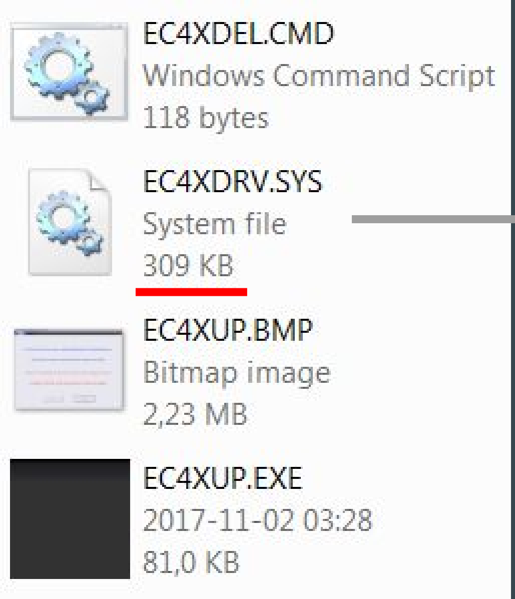
Maybe a similar laptop model?

Portégé S100!

EC: Updates



S100_EC4X01_ENU.EXE



EC4XDEL.CMD
Windows Command Script
118 bytes

EC4XDRV.SYS
System file
309 KB

EC4XUP.BMP
Bitmap image
2,23 MB

EC4XUP.EXE
2017-11-02 03:28
81,0 KB

Inside: 3 update blobs
(different versions)

EC: Update installer

Uses ports 62h & 66h

Sends the 1st part (~2,5KB)

Sends the 2nd part (~100KB)

EC: Update blob

It’s decoded inside EC - no code available :(

Let’s try some analysis!

EC: Update blob - analysis

High entropy ⇒ encryption or compression

No regularities in trigrams ⇒ encryption

Size always divisible by 8 ⇒ encryption

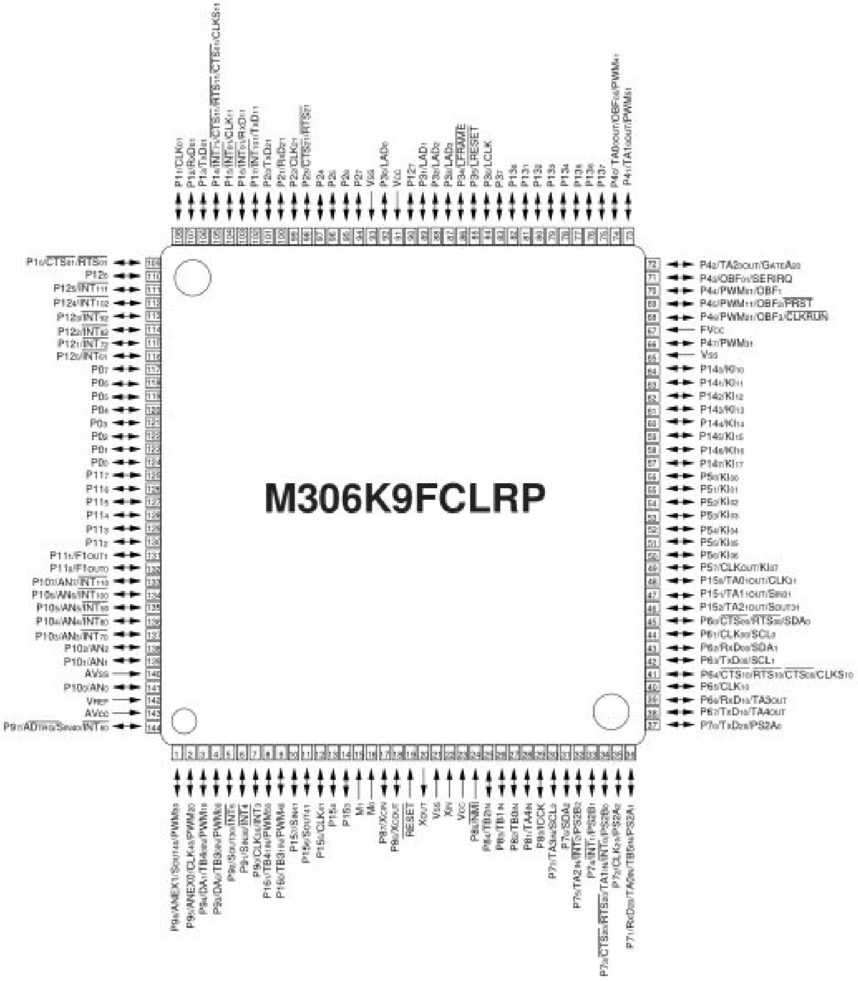
Longest repeated substring is short ⇒ if encryption, then not ECB

EC: Update blob - analysis

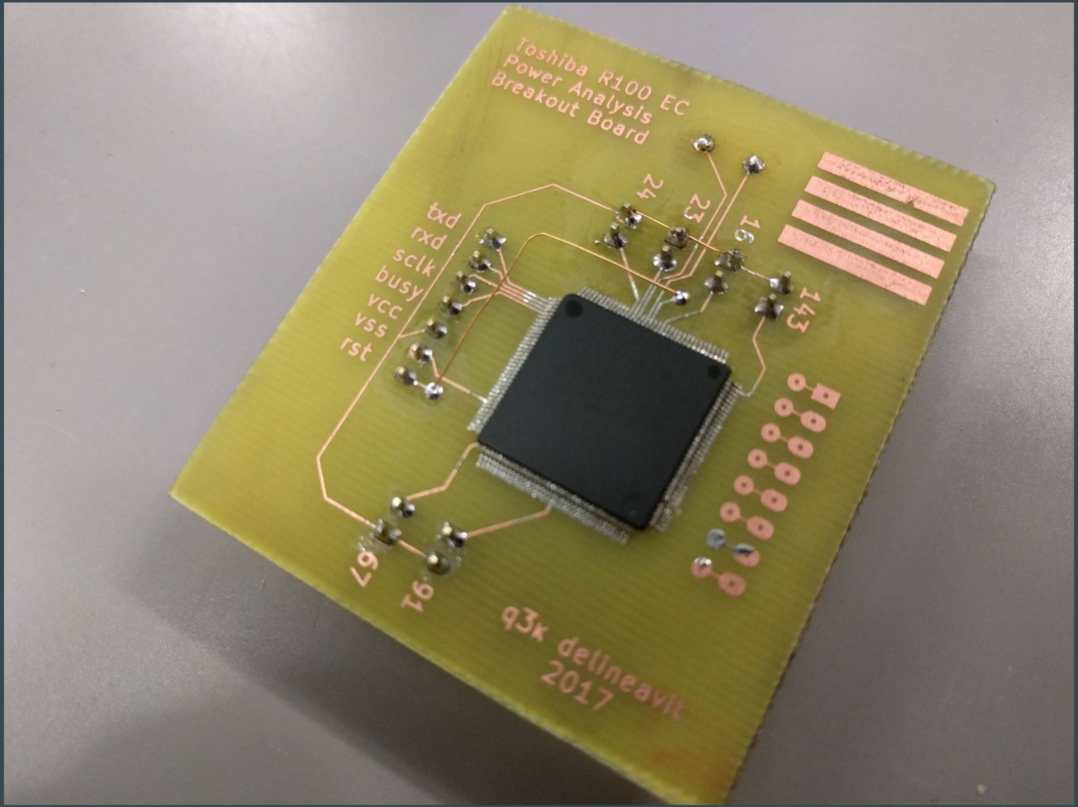
Looks like a dead-end...

Serge, could you please desolder something again...?

EC..?

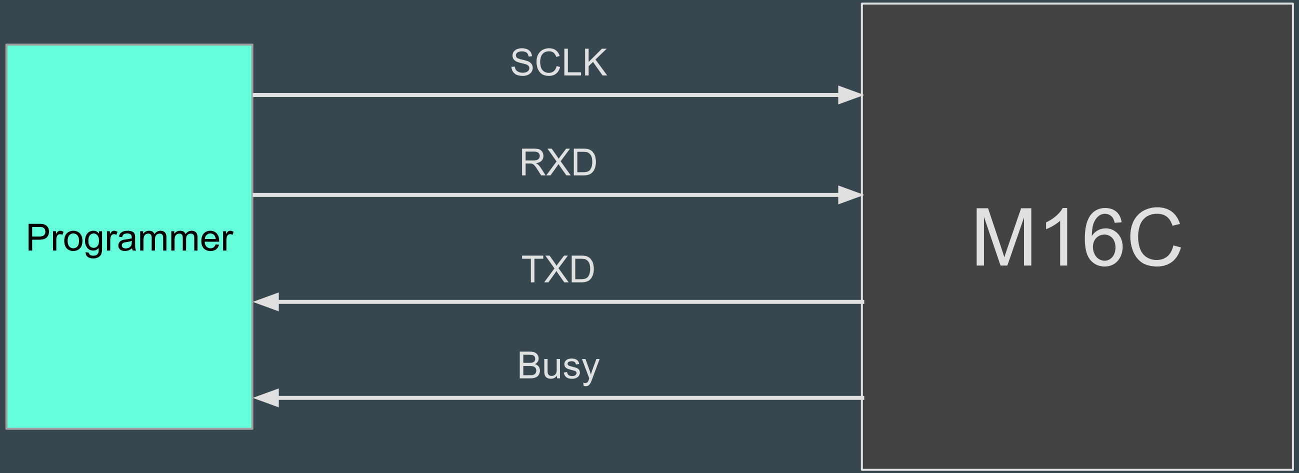

M306K9FCLRP
P1₀/CTS01/RTS01
P12₆
P12₅/INT111
P12₄/INT102
P12₃/INT92
P12₂/INT82
P12₁/INT72
P12₀/INT61
P0₇
P0₆
P0₅
P0₄
P0₃
P0₂
P0₁
P0₀
P11₇
P11₆
P11₅
P11₄
P11₃
P11₂
P11₁/F1OUT1
P11₀/F1OUT0
P10₇/AN7/INT110
P10₆/AN6/INT100
P10₅/AN5/INT90
P10₄/AN4/INT80
P10₃/AN3/INT70
P10₂/AN2
P10₁/AN1
AVSS
P10₀/AN0
VREF
AVCC
P9₇/ADTRG/SIN40/INT60
P4₂/TA2OUT/GATEA20
P4₃/OBF01/SERIRQ
P4₄/PWM01/OBF1
P4₅/PWM11/OBF2/PRST
P4₆/PWM21/OBF3/CLKRUN
FVCC
P4₇/PWM31
VSS
P14₀/KI10
P14₁/KI11
P14₂/KI12
P14₃/KI13
P14₄/KI14
P14₅/KI15
P14₆/KI16
P14₇/KI17
P5₀/KI00
P5₁/KI01
P5₂/KI02
P5₃/KI03
P5₄/KI04
P5₅/KI05
P5₆/KI06
P5₇/CLKOUT/KI07
P15₀/TA01OUT/CLK31
P15₁/TA11OUT/SIN31
P15₂/TA21OUT/SOUT31
P6₀/CTS00/RTS00/SDA0
P6₁/CLK00/SCL0
P6₂/RxD00/SDA1
P6₃/TxD00/SCL1
P6₄/CTS10/RTS10/CTS00/CLKS10
P6₅/CLK10
P6₆/RxD10/TA3OUT
P6₇/TxD10/TA4OUT
P7₀/TxD20/PS2A0

One last breakout later…

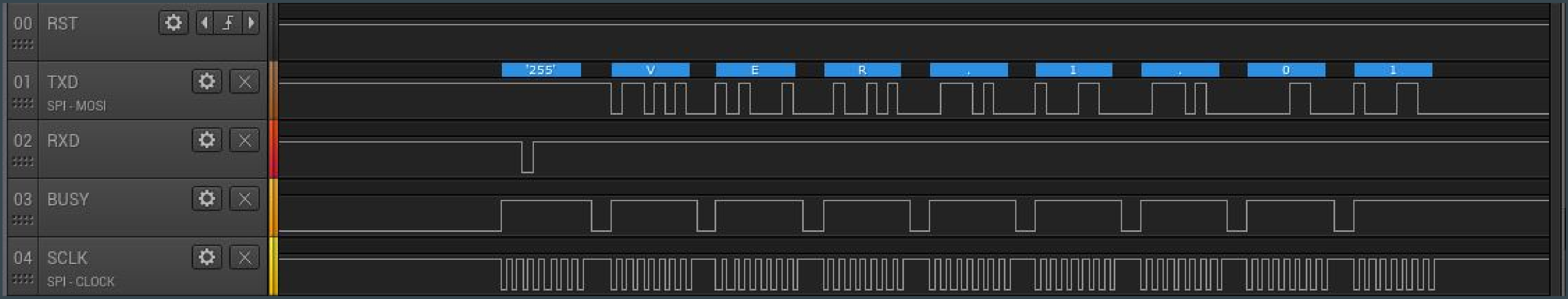


Let’s dump this thing.

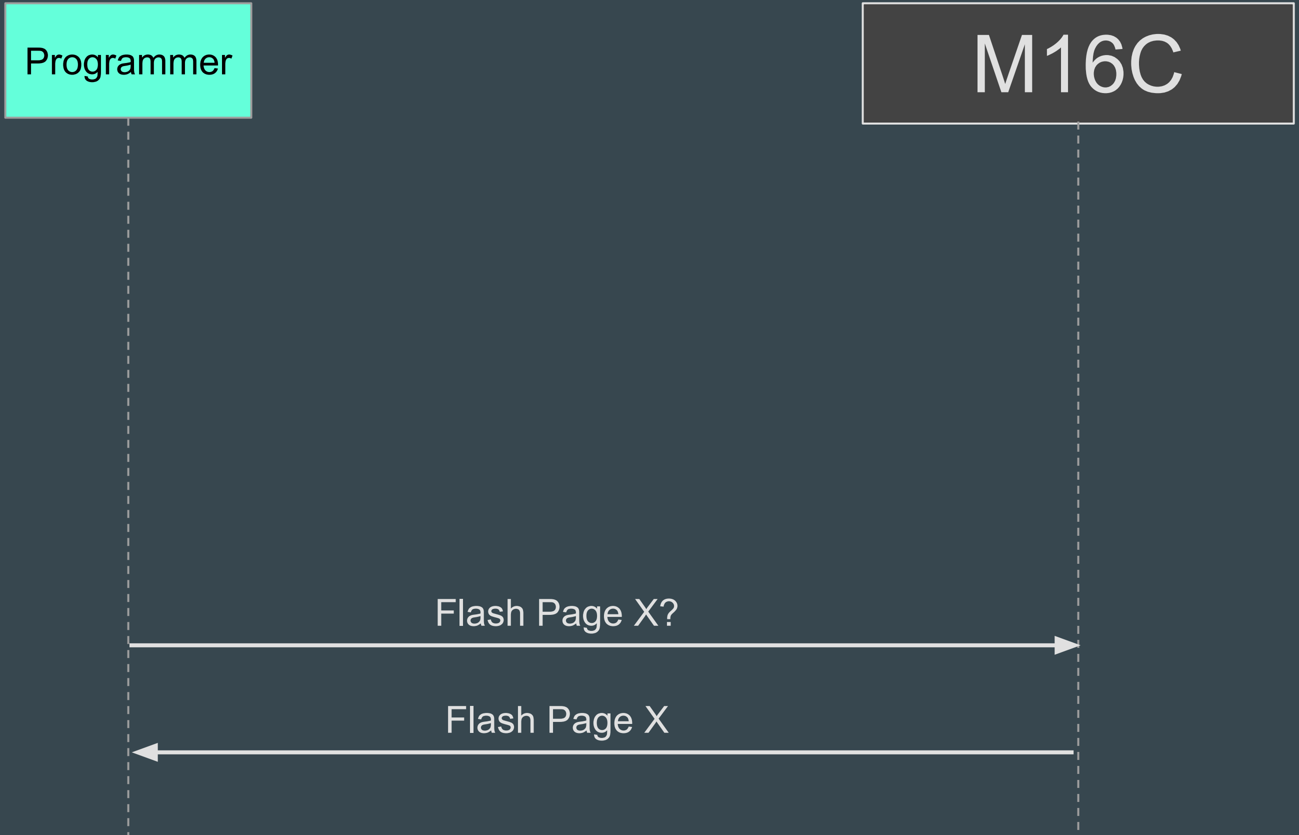
EC: Programming Protocol



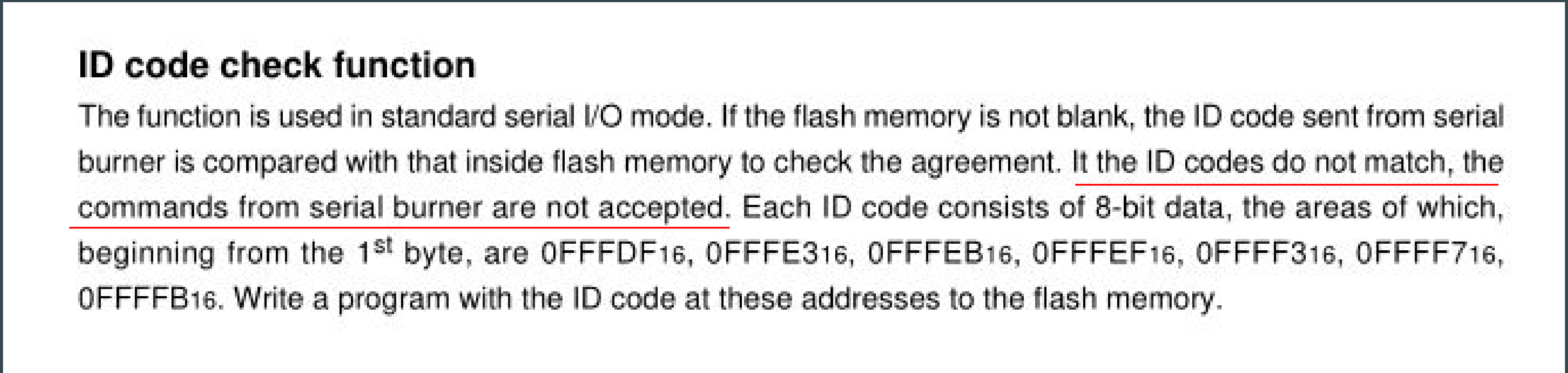
EC: Programming Protocol



EC: Programming Protocol



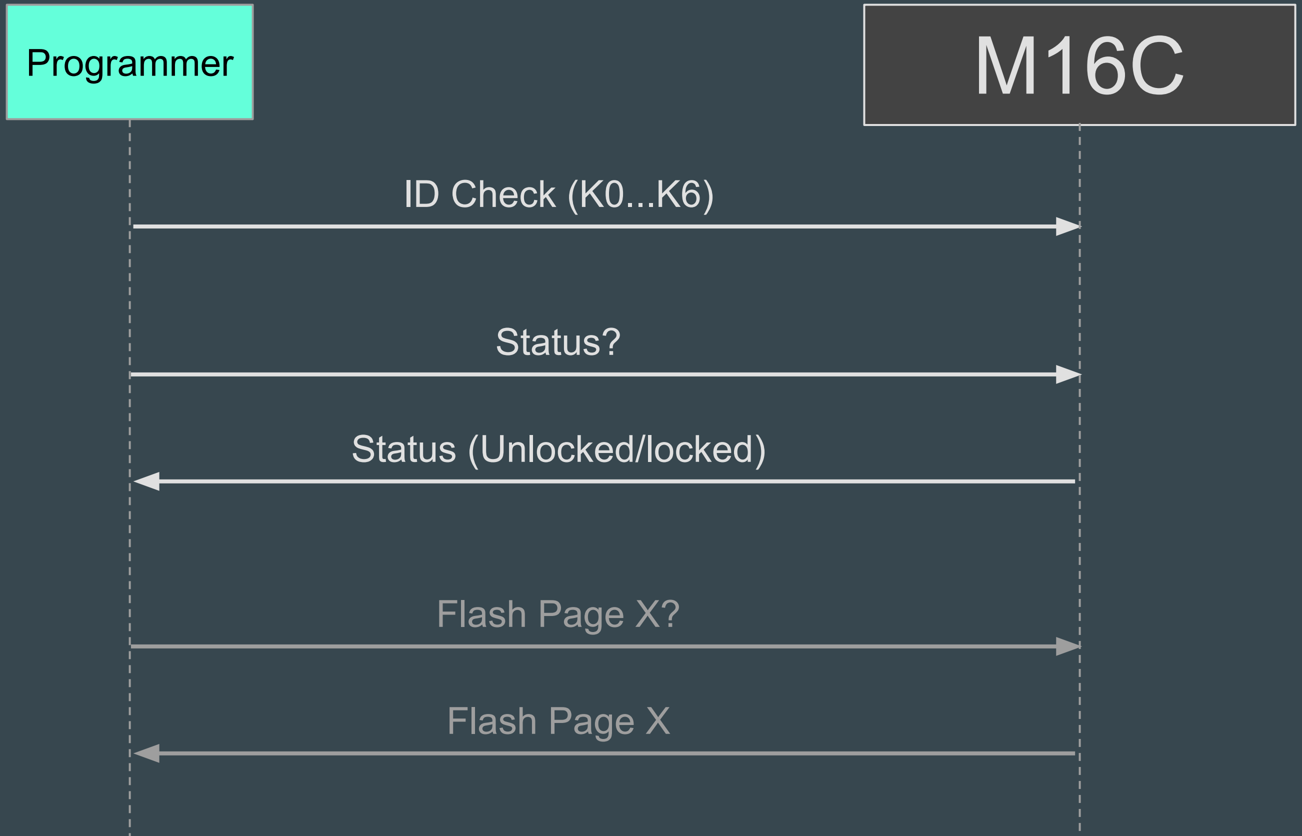
Not so fast



ID code check function

The function is used in standard serial I/O mode. If the flash memory is not blank, the ID code sent from serial burner is compared with that inside flash memory to check the agreement. It the ID codes do not match, the commands from serial burner are not accepted. Each ID code consists of 8-bit data, the areas of which, beginning from the 1st byte, are $0FFFDF_{16}$, $0FFFE3_{16}$, $0FFFEB_{16}$, $0FFFEF_{16}$, $0FFFF3_{16}$, $0FFFF7_{16}$, $0FFFFB_{16}$. Write a program with the ID code at these addresses to the flash memory.

EC: Programming Protocol



Programmer

M16C

ID Check (K0...K6)

Status?

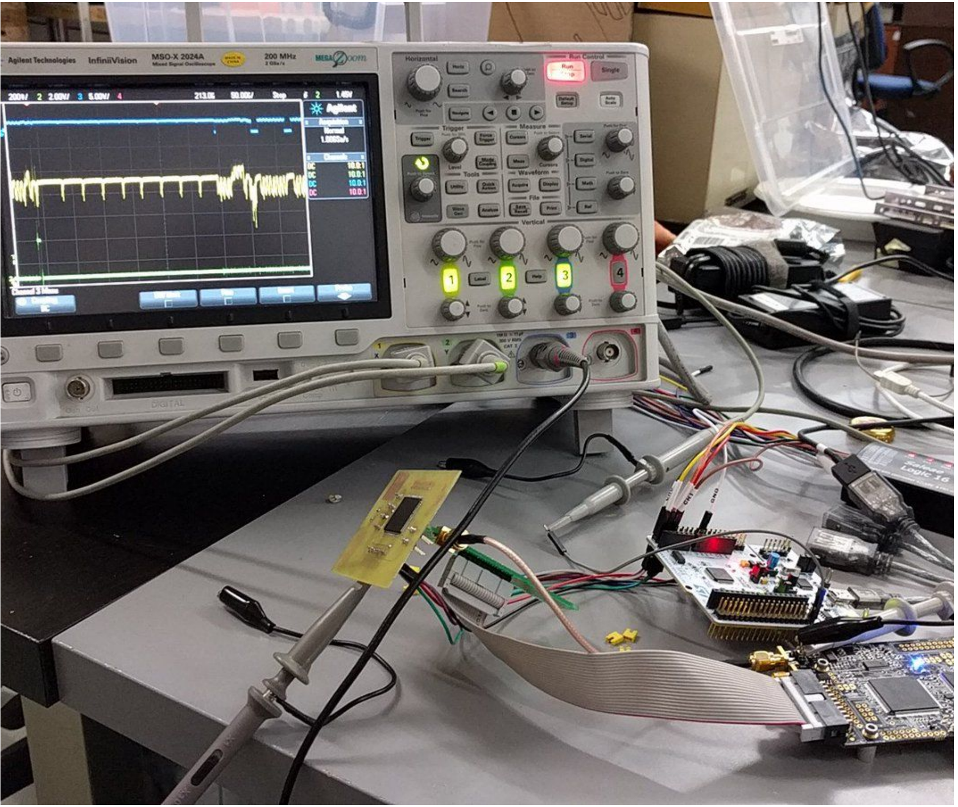
Status (Unlocked/locked)

Flash Page X?

Flash Page X

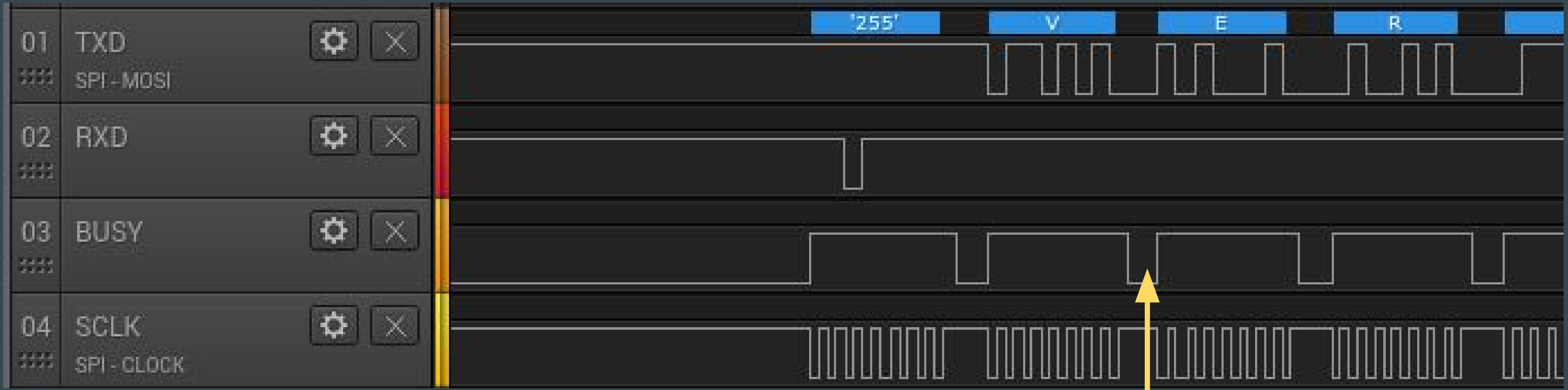
Side channel attacks?

Fault injection?



Not so fast.

Software level ‘side’ channels



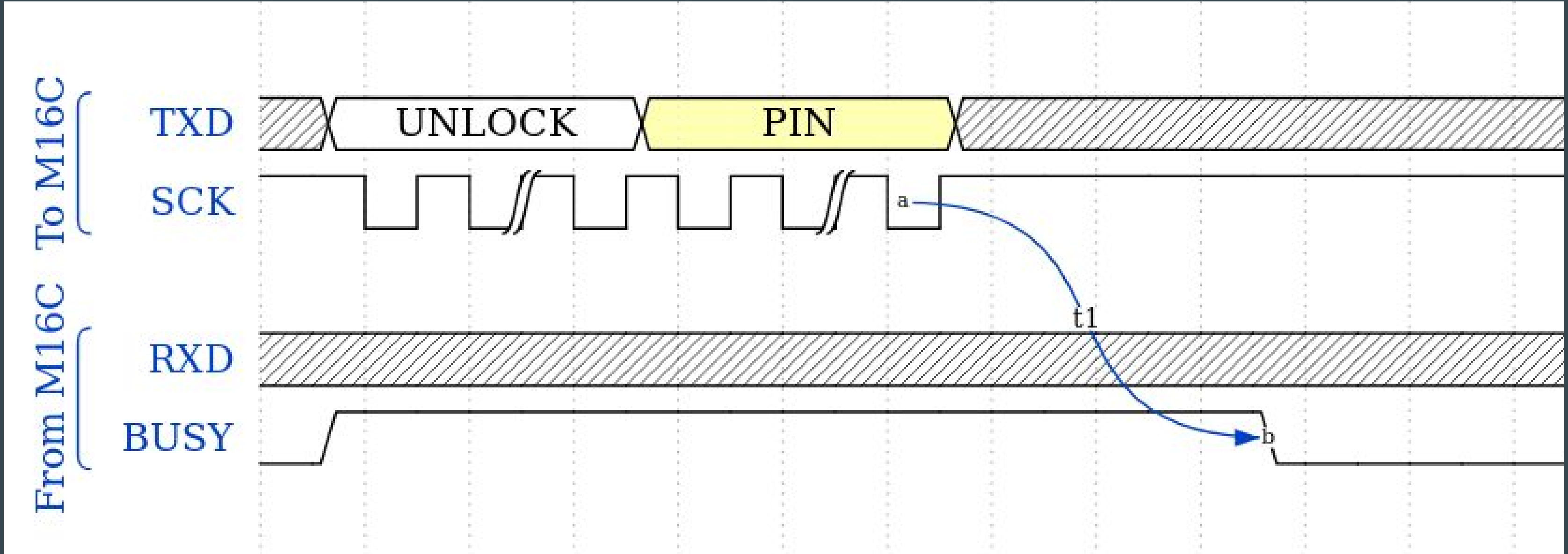
An PIN unlock request does not result in any immediate success/failure transmission, but...

Hmm.

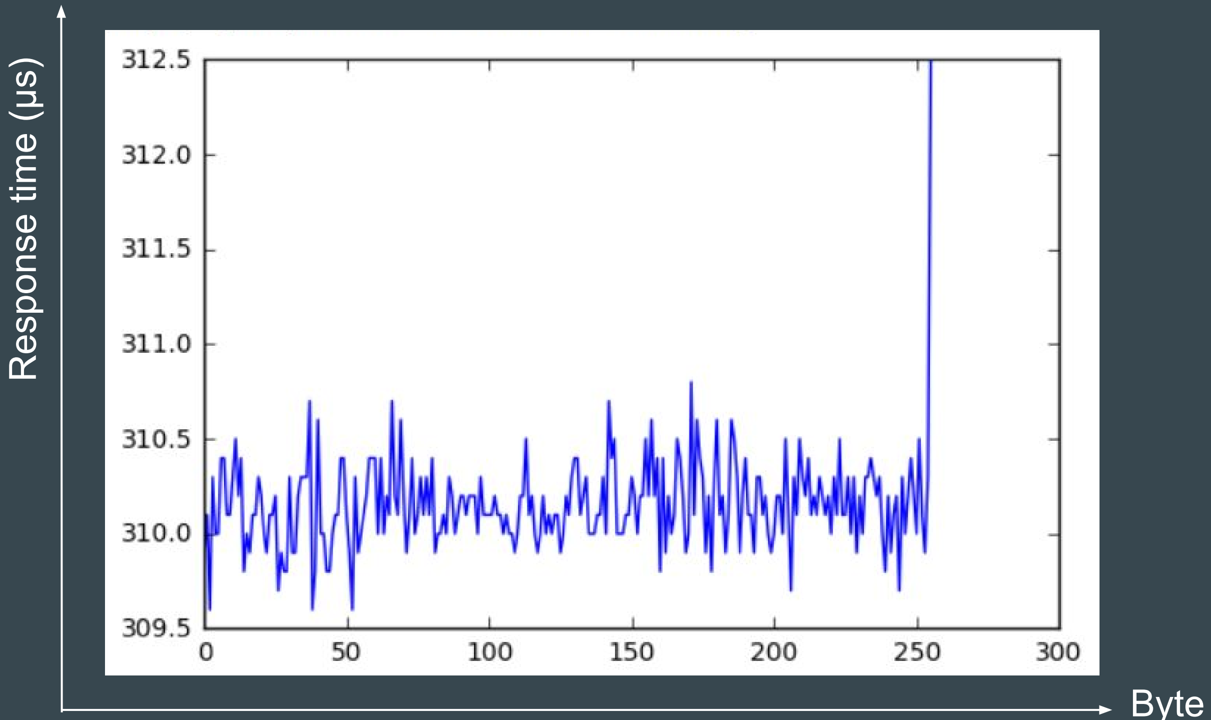
EC: M16C bootloader bug

Let’s run some quick tests.

EC: M16C bootloader bug

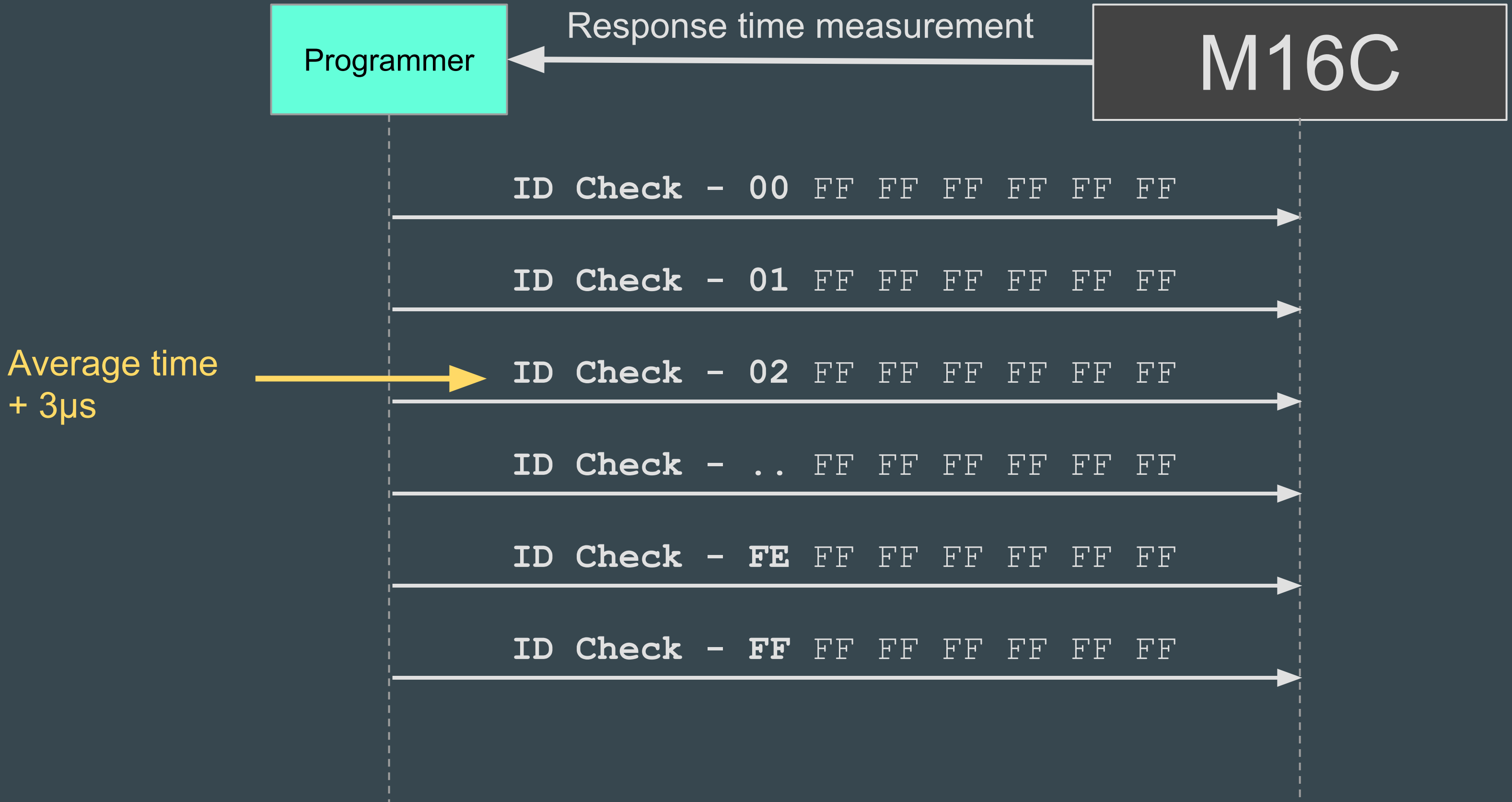


EC: M16C bootloader bug



Well that’s not good.

EC: M16C Bootloader bug



Programmer

Response time measurement

M16C

ID Check - 00 FF FF FF FF FF FF

ID Check - 01 FF FF FF FF FF FF

ID Check - 02 FF FF FF FF FF FF

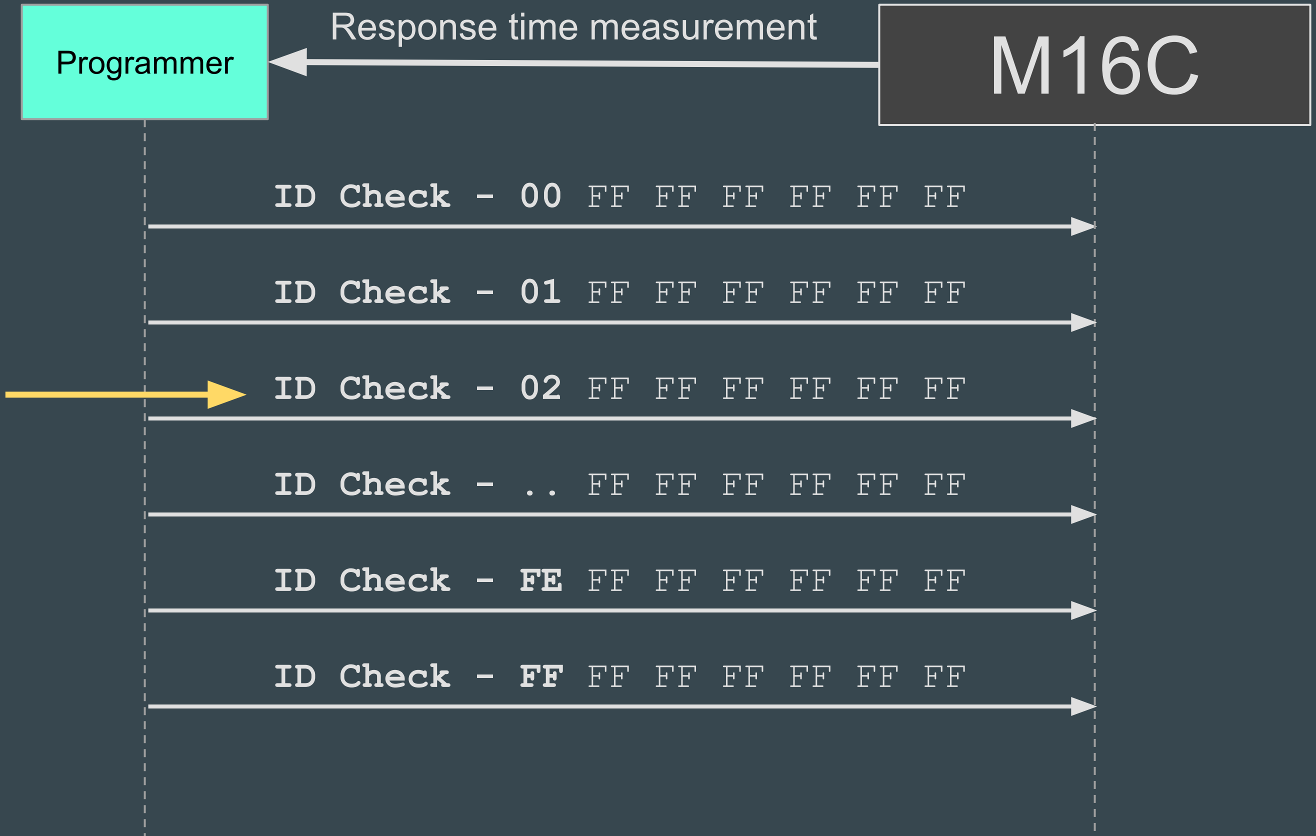
ID Check - .. FF FF FF FF FF FF

ID Check - **FE** FF FF FF FF FF FF

ID Check - **FF** FF FF FF FF FF FF

Average time
+ 3µs

EC: M16C Bootloader bug



Programmer

Response time measurement

M16C

ID Check - 00 FF FF FF FF FF FF

ID Check - 01 FF FF FF FF FF FF

ID Check - 02 FF FF FF FF FF FF

ID Check - .. FF FF FF FF FF FF

ID Check - FE FF FF FF FF FF FF

ID Check - FF FF FF FF FF FF FF

Average time + 3µs

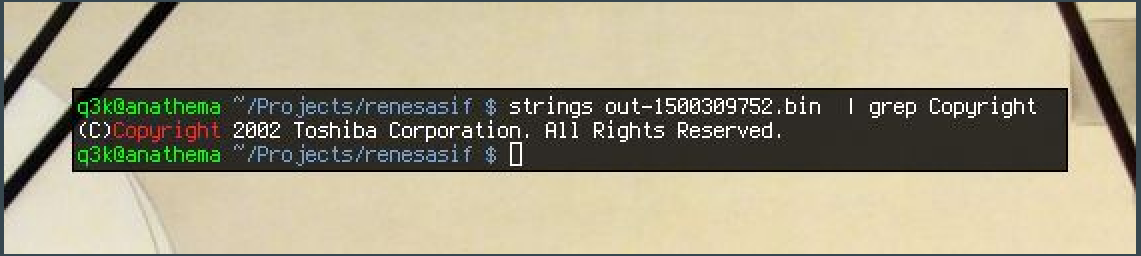
Ergo, the first byte of the key is 02.

EC: M16C Bootloader bug

Thus, we can enumerate all bytes of the key one by one, using the timing difference for each correct byte to reduce our search to just 0x100*7 checks.

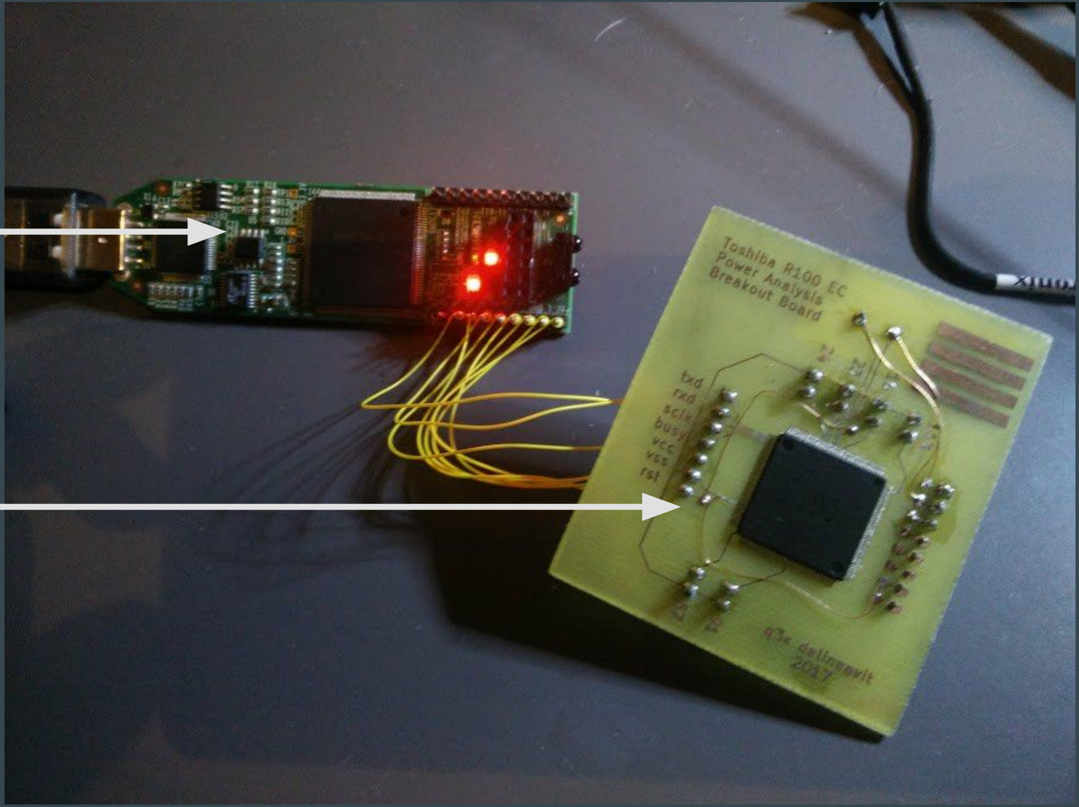
And we get the key.

EC: M16C Bootloader bug



```
q3k@anathema ~/Projects/renesasif $ strings out-1500309752.bin  | grep Copyright
(C)Copyright 2002 Toshiba Corporation. All Rights Reserved.
q3k@anathema ~/Projects/renesasif $ 
```

EC: M16C Bootloader bug



FPGA
(iCE40)

(EC)
M16C

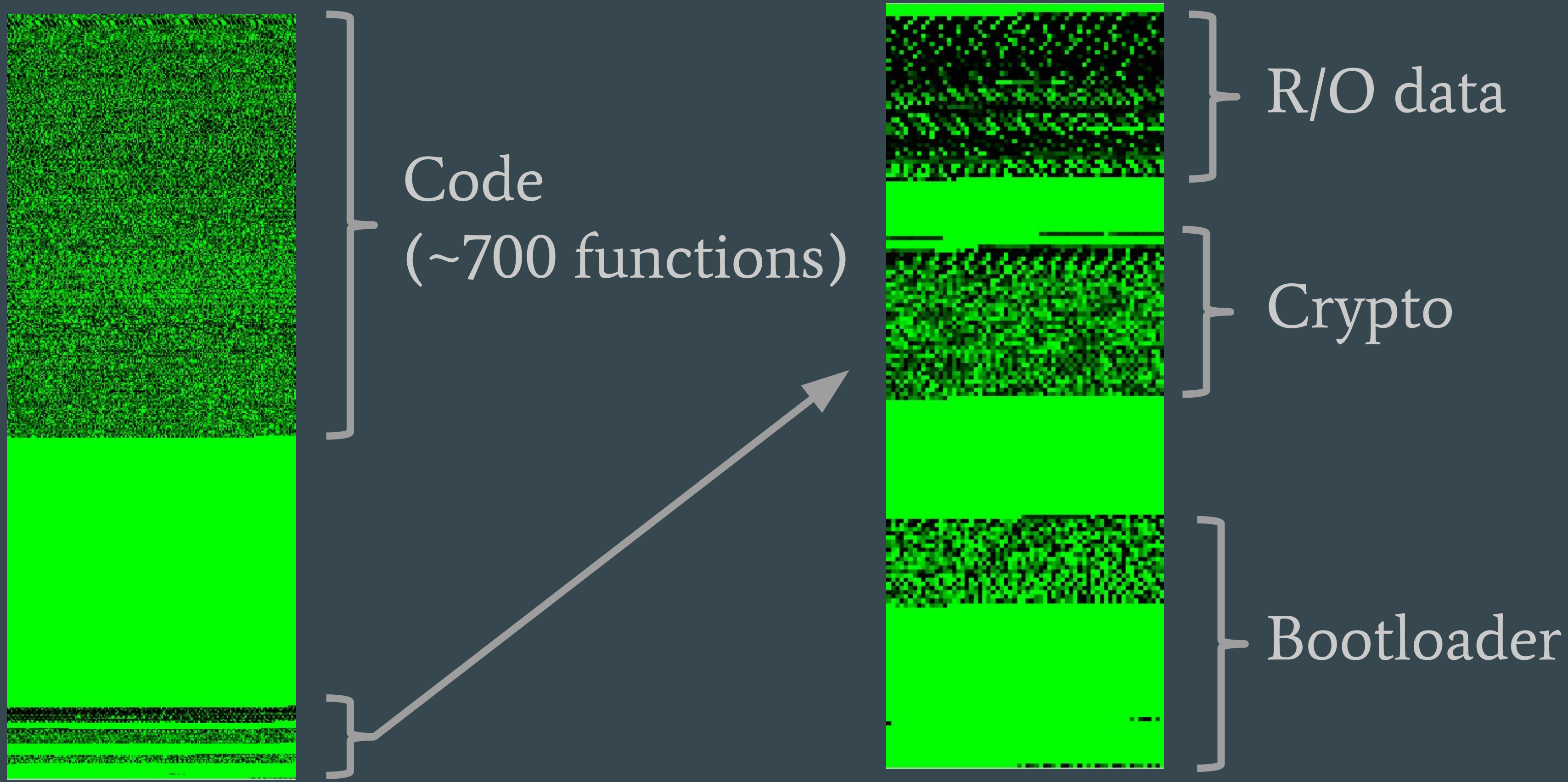
EC: M16C Bootloader bug

PoC || GTFO

https://github.com/q3k/m16c-interface/

(note: doesn’t work for all M16Cs... yet)

EC: RE



Code
(~700 functions)

R/O data

Crypto

Bootloader

EC: RE

Much simpler code than in the BIOS

No strings

We’re looking for LPC communication and BIOS-call table

EC: RE

Finding the table is easy

~100 different BIOS<->EC calls

We know the numbers of the interesting calls ⇒ let’s analyze the handlers!

Sounds easy...?

EC: RE of the handlers

Manual context-switching

No common call convention

Handlers aren’t split into functions

Jumps to the middle of other functions

Password check: BIOS

```
out_buf = call_EC(
    func=0x24,
    in_buf=MD5(input)[:8] + pwd_type
)
```

```
out_buf[0] == 0 ⇒ success
```

Password check: EC

Let’s look at the handler on the EC side…

…6 levels down the call hierarchy:

```
BMGEU/C p6_4, p6
BSET    pd6_4, pd6
JSR.W   set_p6_5
JSR.W   clear_p6_5
```

I/O on pins 40 & 41

Password check: EC

Oh, come on... :(

Password check: EC

This time it's only an EEPROM :)

EC reads one block, decrypts it and compares with the received MD5

Challenge/Response

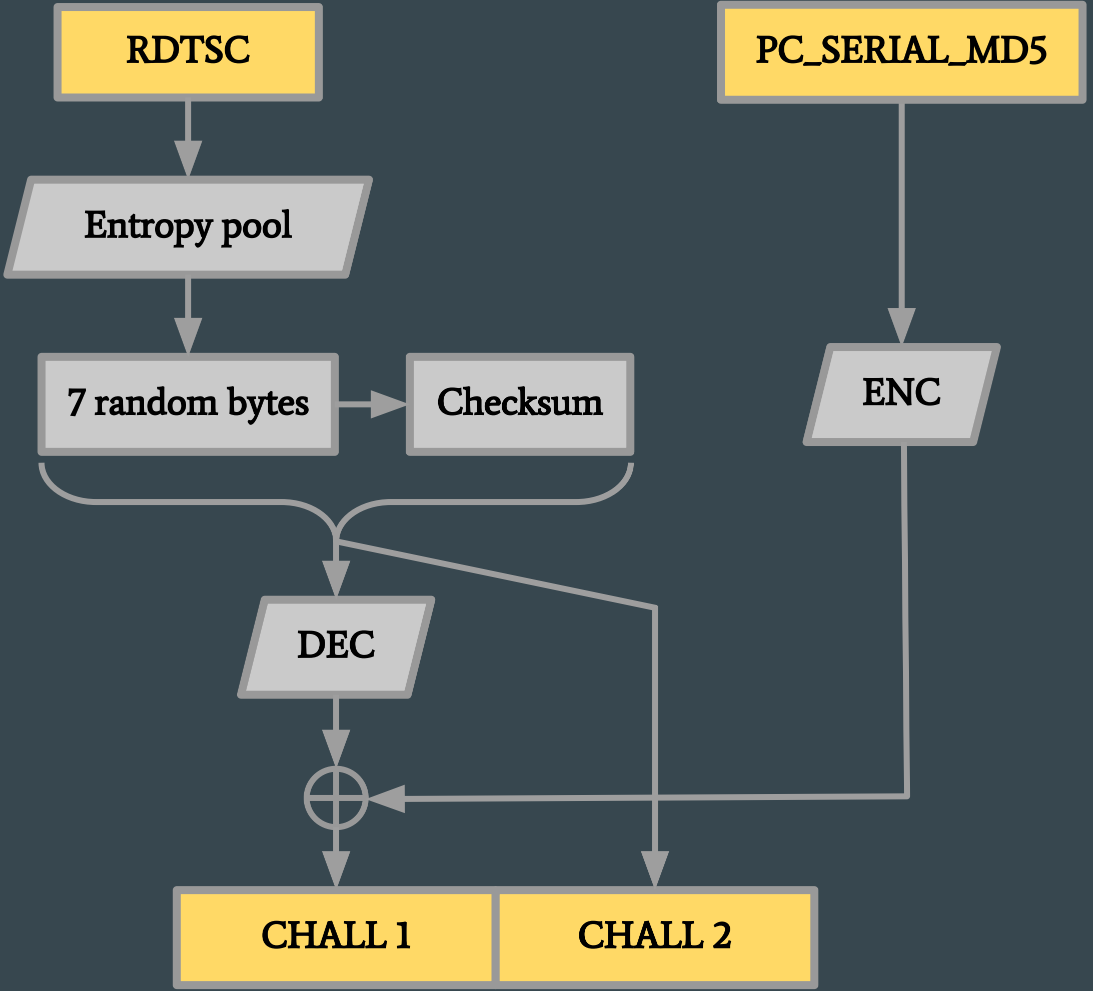
Screw it, we’re looking for a universal attack

Let’s look at the challenge/response!

Challenge: BIOS

```
out_buf = call_EC(
    func=0x1A,
    in_buf=rdtsc() + MD5(pc_serial)[:8]
)
challenge = bytes_to_string(out_buf)
```

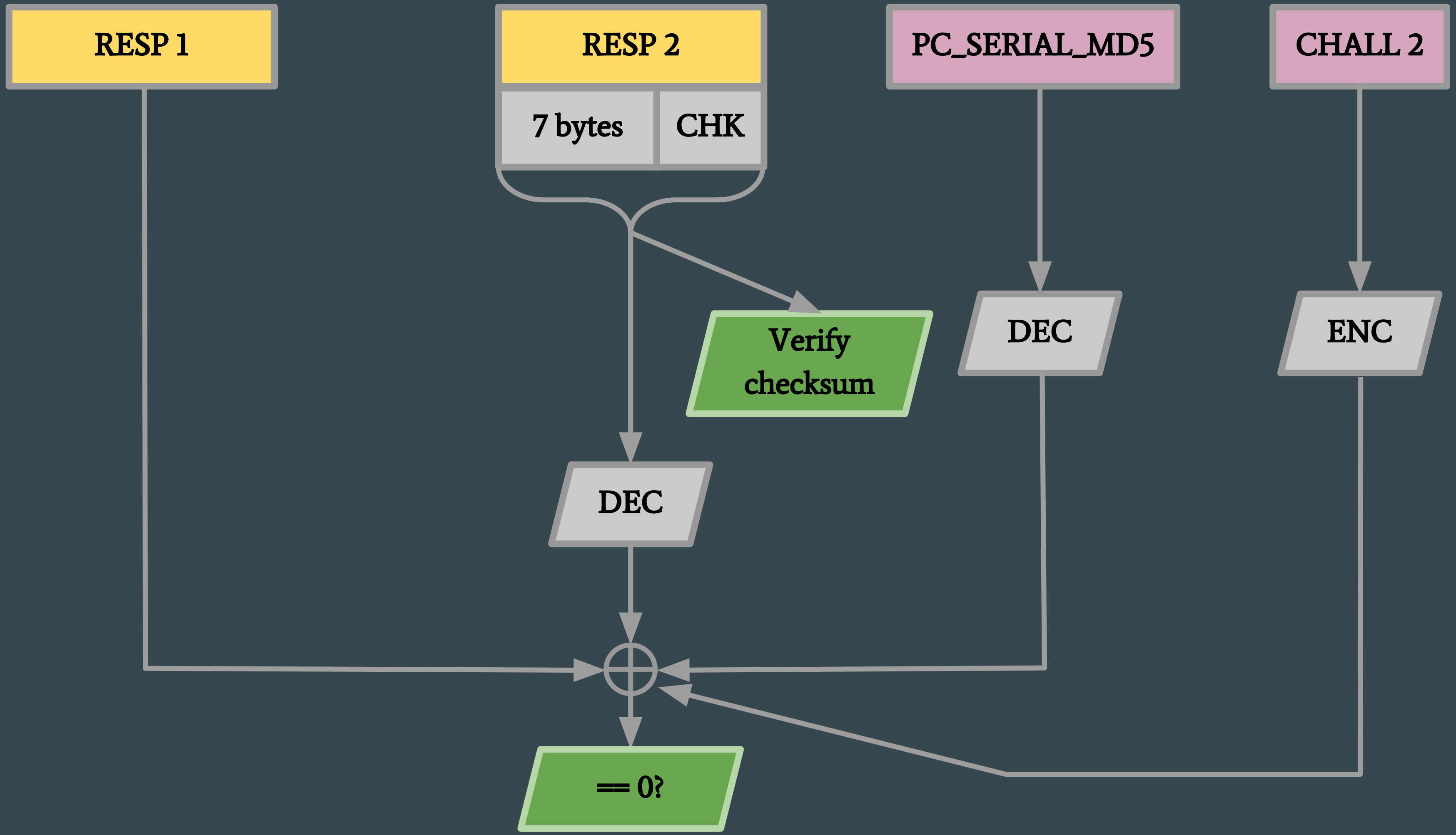
Challenge: EC



Response: BIOS

```
out_buf = call_EC(
    func=0x1B,
    in_buf=string_to_bytes(user_input)
)
```

`out_buf[0]` ⇒ success/fail

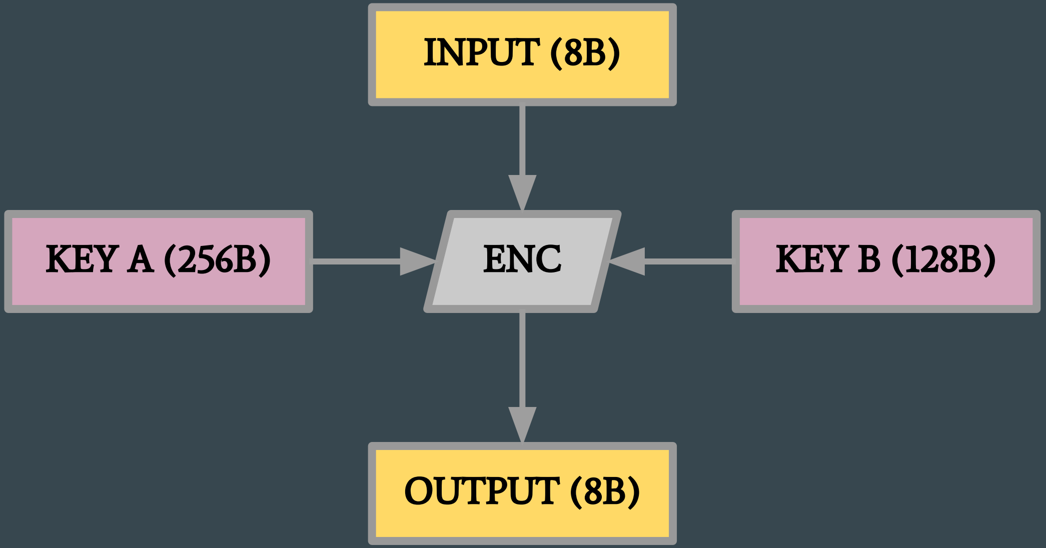

RESP 1
RESP 2
7 bytes
CHK
PC_SERIAL_MD5
CHALL 2
Verify checksum
DEC
DEC
ENC
== 0?

EC: Encryption

ENC? DEC?

EC: Encryption

A custom 64-bit block cipher



Challenge/Response

We just need to rewrite it in Python and …

DEMO!

EC: Update system

Let’s decrypt the updates!

EC: Update system

Uh, symmetric signatures?

We can generate our own!

So, how's it like on their newer laptops?

If it ain’t broke, don’t fix it!

(that applies to keys, too)

Impact

Unlocking any (business) laptop.

Permanent rootkit in the EC.

We can attack the host from the EC.

Rootkit in EC?

DMA to the host via LPC (not supported by this particular EC) .

Keylogging & storage.

USB-Rubber-Ducky-like (key/mouse injection).

BIOS exploitation via the internal API.

Official Toshiba statement (from 2017-11-02)

Toshiba is working on a temporary BIOS update that can be used to prevent the security issue that has been raised and expects to release this update on its website within the next 2 weeks.

Toshiba plans to start the release of a permanent fix for some models from January, 2018 and will complete the releases of permanent fix for all applicable models by the end of March 2018.

Questions?

https://q3k.org/slides-recon-2018.pdf