Lost your “secure” HDD PIN? We can help!

Julien Lenoir & Raphaël Rigo (firstname.name@airbus.com)

H2HC - 2016-10-23

About us

We work for Airbus Group Innovations’ cybersecurity lab (TX4CS).

Raphaël Rigo

- reverser
- interested in low-level stuff
- `https://syscall.eu`

Julien Lenoir

- reverser
- interested in vulnerability research
- main activity: security assessment on various products

Today

Zalman ZM-SHE500

Zalman ZM-VE500

Previous work

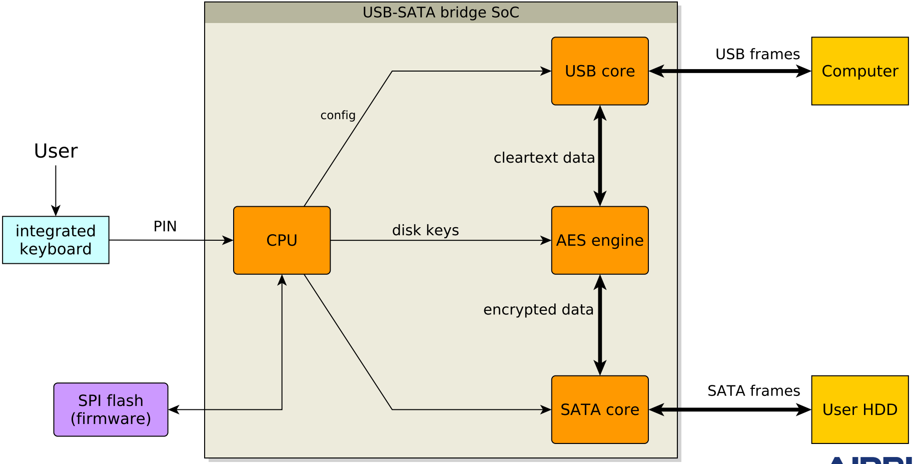
HDDs with hardware keyboard

- Spritesmods [Dom10] :
  - iStorage diskGenie PIN bruteforce with timing attack
- Colin O’Flynn [O’F16] :
  - LockDown PIN bruteforce and side channels
- Czarny & Rigo [CR15] :
  - Zalman ZM-VE400 circuits and logic reversing

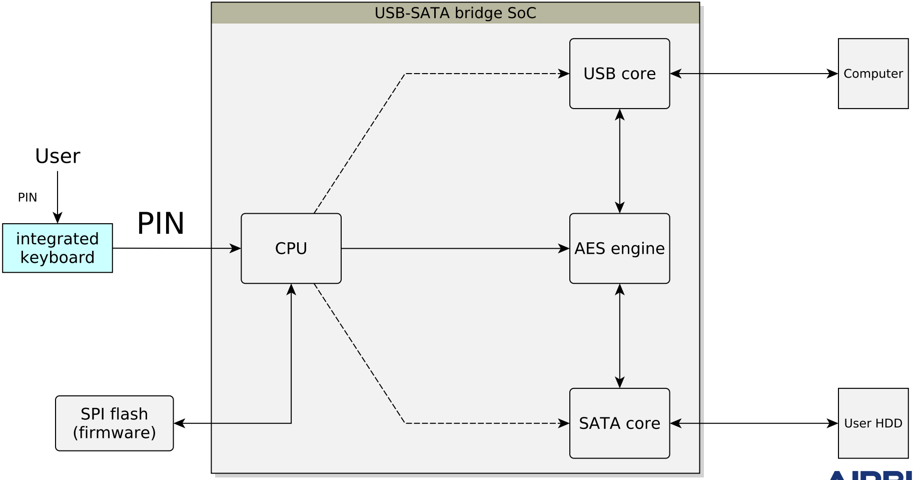
HDDs with software unlock

- “got hw crypto?” Alendal, Kison, modg [AKm15] :
  - Western Digital crypto fails and backdoors

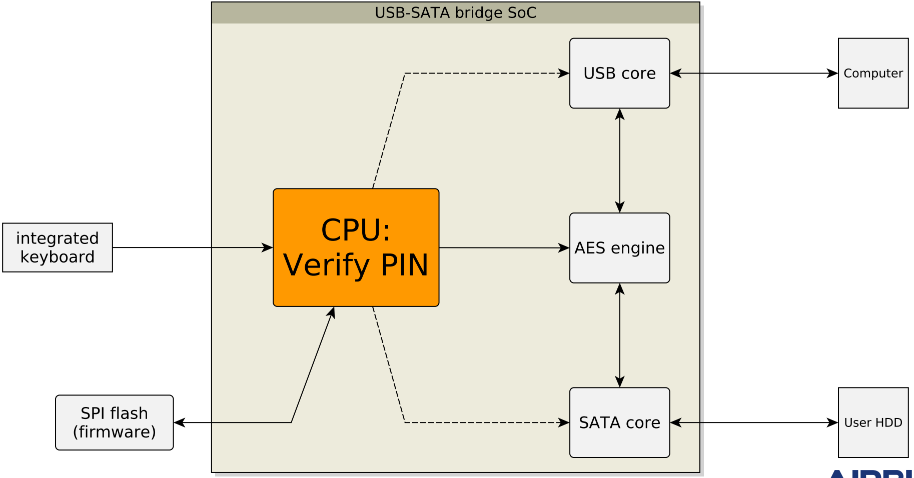
Overall architecture



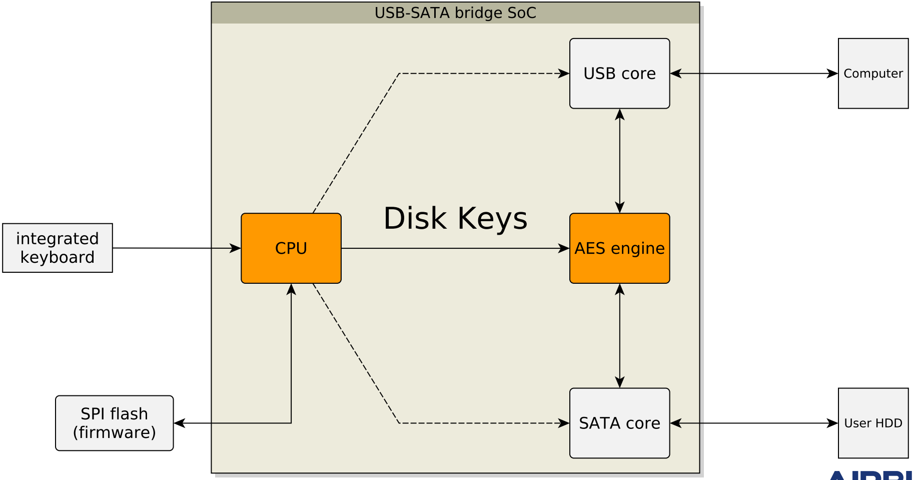
Basics: Unlocking a drive. (1) Entering PIN



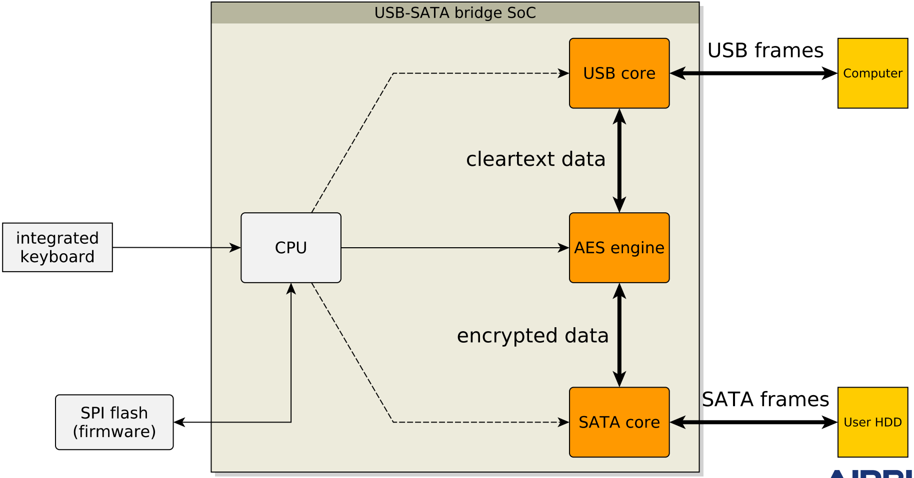
Basics: Unlocking a drive. (2) Verifying PIN



Basics: Unlocking a drive. (3) Configuring encryption



Basics: Unlocking a drive. (4) Accessing data



Characteristics

Data protection: AES-256-XTS

- hardware-implemented for performance
- recognized disk encryption standard (random access + differentiation)
- requires **two** 256-bit keys to encrypt full drive

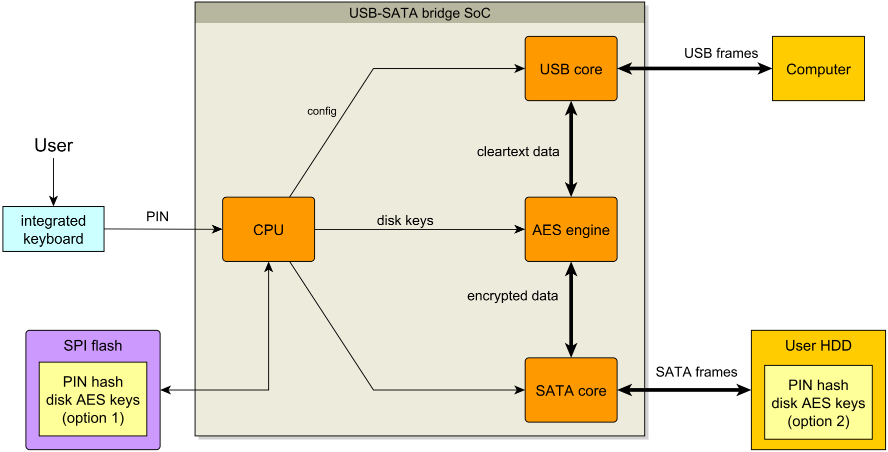
User-friendliness

- tells user if the PIN is right or wrong
- allows PIN change without re-encrypting the whole drive, drive keys never change!

Needs

- secure storage for PIN verification means
- secure random generation of AES keys
- secure storage for AES keys

Storing secrets options



Our approach

Mainly software, no elite hardware skills involved

We want to understand

- how and where are disk keys stored:
  - are they also encrypted?
  - can they be extracted?
- how random disk keys are: can they be brute-forced somehow?
- how PIN is verified: bypass of any kind?

Our goal

Access user files on a stolen/found drive **without** PIN

First steps

Basic crypto testing:

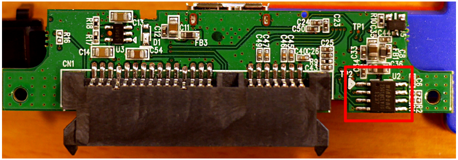
- verify that encryption is actually done:
  - write data using encryption
  - check that data is encrypted using a normal USB-SATA bridge
- verify that the key is not constant or derived directly from the PIN

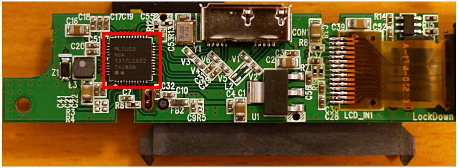
Enclosure test

- verify if the disk is tied to a specific enclosure:
  - configure encryption
  - try to use disk in new enclosure

Zalman ZM-SHE500

Info





Hardware

- MediaLogic MLDU03, really a rebranded Renesas uPD72023 (no data sheet)
  - integrated V850 microcontroller (hard to identify. . .)
- SPI flash
- actually designed by SKYDIGITAL (marking on PCB)

Software

- firmware updater and **unencrypted** updates available

Association and basic testing

Can be associated with up to 50 drives.
Enclosure associated with the drive:

- once PIN is first set, 4 to 8 digits
- master key for rescue purpose

Observations:

- crypto seems OK
- **disk keys NOT stored on drive**, in the flash?

Next step

Reverse engineer firmware and updater

Master key displayed

Updater’s hidden commands

Updater binary has hidden commands:

- MEMDUMPALL
- ROMDUMPALL

Full dump of:

- device RAM
- device SPI Flash

Even on locked drive, before PIN

```
Usage: fwdu03 [option... ]   image-filename
<option> /INFO                  Chip Info.
         /D=n                   Device Index(n=0..9)
         /LIST                  Device List
         /SNTXT                 Use "SN.TXT" file for Serial
         /SNCMDLINE xxxxxx      Use Command Line "xxxxxx" fo
                                Serial Number Length = 1 to
         /UPDATE                F/W Update(Write Only F/W in
         /BINIMG xxxxxx yyyyyy  image-filename
```

Command line

```
                                ; CODE XREF: sub_4075C5+
push    0Ch                     ; MaxCount
push    offset aRomdumpallf ; "ROMDUMPALLF="
push    esi                     ; Str1
call    edi ; _strnicmp
add     esp, 0Ch
```

Hidden command

Cool backdoor

How it works:

- constructor specific SCSI commands over USB
- example: `0xFD` to dump RAM

Talked with the supplier:

- feature/backdoor in MediaLogic chip
- no patch possible!

We used it to:

- dump SPI flash content, looking for secrets
- dump RAM to help reverse engineering the firmware
- avoid soldering on the board :)

Flash content

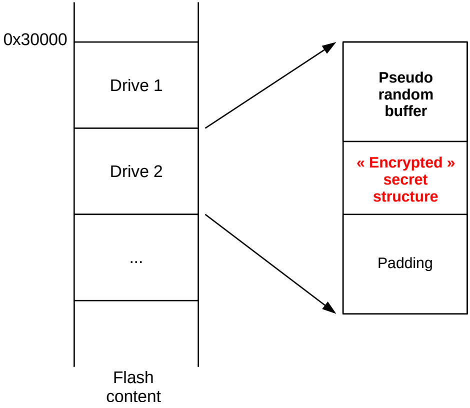
Interesting blobs:

- stored at `0x30000`
- one per associated drive

Composed of:

- two random buffers
- one `0x90` bytes encrypted-like structure

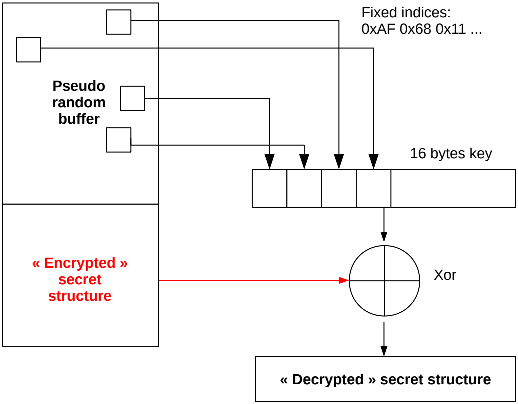
Disk keys stored in this structure?



Let’s decode it

Basically just encoded:

- construct 16 bytes key from pseudo-random buffer
- repeatedly *xor* secret structure



Secret structure content

Once decoded:

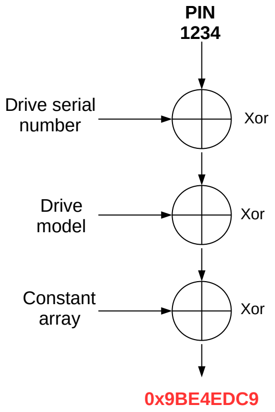
- drive model
- drive serial number
- weird integers:
  - `0x006ACFE7`: timestamp
  - `0x9BE4EDC9`: current PIN
  - `0x9B7F7D59`: initial PIN

No random vectors. . . **no disk keys?**

```
53 31 30 55 4A 44 30 50 38 32 36 37 31 35 20 20   S10UJD0P826715
20 20 20 20 53 41 4D 53 55 4E 47 20 48 4D 31 36       SAMSUNG HM16
30 48 49 20 20 20 20 20 20 20 20 20 20 20 20 20   0HI
20 20 20 20 20 20 20 20 20 20 20 20 E7 CF 6A 00               çÏj.
59 7D 7F 9B 53 31 30 55 4A 44 30 50 38 32 36 37   Y}.›S10UJD0P8267
31 35 20 20 20 20 20 20 53 41 4D 53 55 4E 47 20   15      SAMSUNG
48 4D 31 36 30 48 49 20 20 20 20 20 20 20 20 20   HM160HI
20 20 20 20 20 20 20 20 20 20 20 20 E7 CF 6A 00               çÏj.
C9 ED E4 9B 00 03 00 17 1F 27 2F 37 41 49 51 59   Éíä›...
```

Secret structure content

PIN verification algorithm



Steps

- PIN:
  - 0-pad
  - convert to integer
- xor with: model, S/N and constant array

Collisions

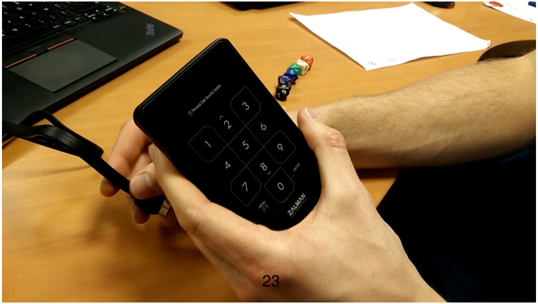
- due to integer conversion of PIN
- collisions for 1234:
  - 12339
  - 123389
  - 1233889
  - 12338889

Attack scenario

With physical access to a powered-off drive like in a hotel room.

So we can:

- dump flash with SCSI commands before authentication
- decode secret structure to get encoded PIN
- finally recover PIN value :)



Cool, but what about disk keys?

Still do not know where and how disk keys are stored.
Reversed engineered further:

- located initialization of AES engine
- *memcpy* of keys to MMIO
- keys are taken from RAM
- where a copy of the secret structure is stored

Disk keys are **really** in secret structure.

```
movea   0x20, r28, r29
mov     0x3FE2410, r6     -- MMIO base
ld.bu   -0x759A[gp], r8
mov     r28, r7
jarl    memcpy_, lp
mov     0x3FE2430, r6     -- MMIO + 0x20
ld.bu   -0x759A[gp], r8
mov     r29, r7
jarl    memcpy_, lp
```

Chip MMIO init

Right before our eyes

Keys made of:

- time dependent value: 4 bytes
- first PIN encoded: 4 bytes
- drive model and S/N: 56 bytes

- first key:

Time dependent value | First PIN | Drive S/N + model

```
E7 CF 6A 00 59 7D 7F 9B 53 31 30 55 4A 44 30 50   çÏj.Y}.›S10UJD0P
38 32 36 37 31 35 20 20 20 20 20 20 53 41 4D 53   826715      SAMS
```

- second key:

Drive S/N + model

```
55 4E 47 20 48 4D 31 36 30 48 49 20 20 20 20 20   UNG HM160HI
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
```

Offline drive attack

Theory

Attacker can bruteforce PIN even without enclosure:

- drive model and serial number are written on the drive
- PIN has less than 32 bits of entropy
- time dependent value can be reasonably reduced to 16 bits

Practice

- brute force in C with OpenMP: 2.5s per timestamp.
- should be broken in less than 24h on a single PC

To sum up

Many issues

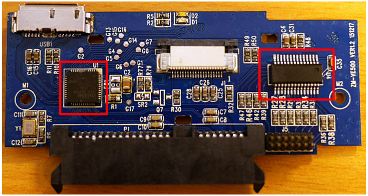
- backdoor in the MediaLogic SoC
- disk keys:
  - weak storage, **updated in new version of firmware**
  - low entropy, keys are predictable
- firmwares are not encrypted nor signed

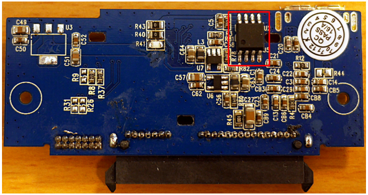
Two attacks

- with enclosure: direct bypass of PIN
- with drive only: recovering disk keys in 24h

Zalman ZM-VE500

Info





Hardware

- Initio INIC3607E (No data sheet)
- Pm25L0032 SPI Flash
- capacitive keyboard controller (no markings)

Software

- firmware updater and **unencrypted** updates available

Basic testing

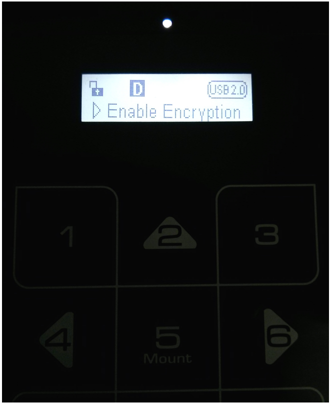
Encryption setup

1. go in menu
2. activate encryption
3. choose PIN between **4** and **8** digits

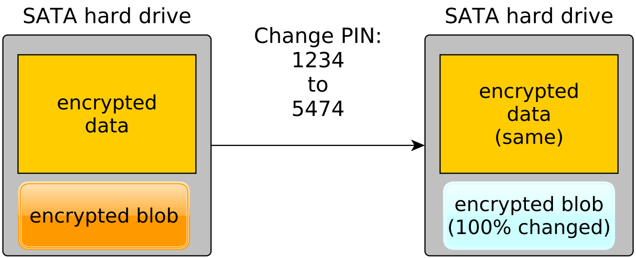
no “master key” displayed

Observations

- crypto seems OK
- drive works in another enclosure



Special blocks on disk



End of drive

- several blocks with a `INI` header: `20 49 4e 49 3a`
- several blocks of high entropy

Leads

Findings

- changing PIN changes the encrypted blob
- disk keys are stored on the drive, probably in the blob

Next step

Reverse the FW to identify how the PIN is verified and where the keys are stored

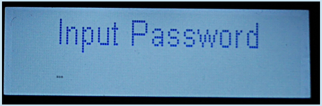
Firmware reversing

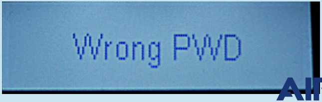
First steps

- search on Google to identify the CPU: **ARCompact**
- spend 1 min to identify loading offset of firmware: `0x4000`
- load in IDA

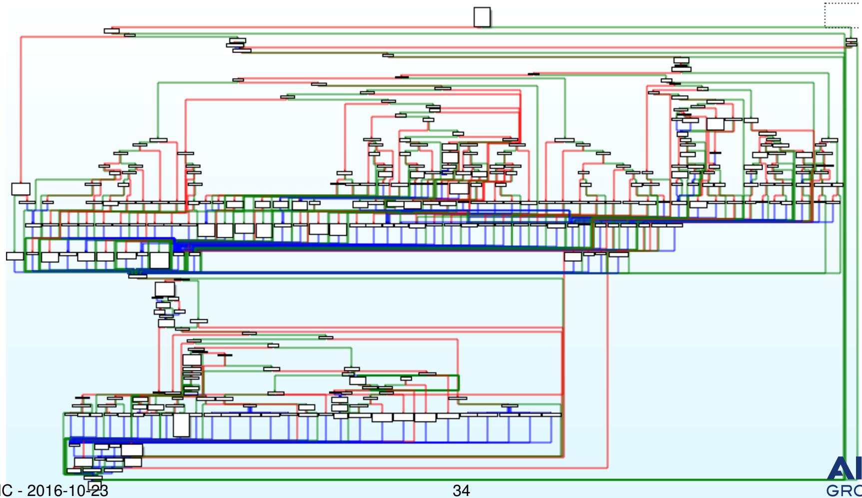
What now?

- we need to find the `check_pin` function, but:
    - no data sheet to identify memory mapped I/O
    - no crypto constants (crypto in HW)
- use strings from LCD!





Menu function



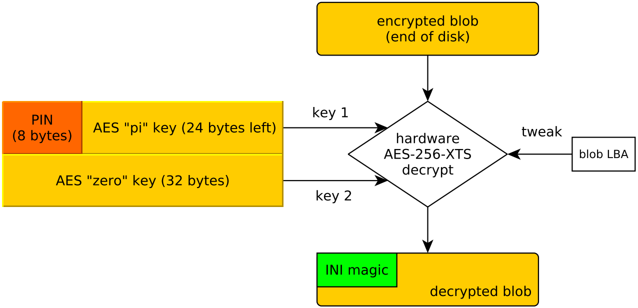
First results

Interesting code around `Wrong PWD`:

- crypto processor MMIO addresses,
- `INI` magic check in a (seemingly) decrypted block
- two weird AES keys (π):

```
Pi_key_256_bits:.byte 3,0x14,0x15,0x92,0x65,0x35,0x89,0x79# 0
                                         # DATA XREF: memcpy_Pi_key_t
                .byte 0x32,0x38,0x46,0x26,0x43,0x38,0x32,0x79# 8
                .byte 0xFC,0xEB,0xEA,0x6D,0x9A,0xCA,0x76,0x86# 0x10
                .byte 0xCD,0xC7,0xB9,0xD9,0xBC,0xC7,0xCD,0x86# 0x18
Pi_key_128_bits:.byte 3,0x14,0x15,0x92,0x65,0x35,0x89,0x79# 0
                                         # DATA XREF: memcpy_Pi_key_t
                .byte 0x2B,0x99,0x2D,0xDF,0xA2,0x32,0x49,0xD6# 8
```

PIN verification algorithm



1. get PIN in 8 byte array, 0 padded
2. `memcpy(aeskey, pin, 8)`: overwrite the start of π key
3. configure HDD crypto engine with AES-256-XTS with:
   - PIN+π as key 1
   - 32 bytes of `0` as key 2
   - sector number as tweak
4. read “secret” block through crypto engine
5. check for magic “`INI`”

PIN 0 padded $\Longrightarrow$ collisions

So, are we done?

So, we can do our bruteforcer, right?

- read secret block
- for each candidate PIN:
  - decrypt
  - check for `INI`

Result

Nothing.

Next step

Reverse more to understand why.

Need for “Debugging”

Problems

- contrary to SHE500, no way of looking at memory
- we would like to interact with the running code
- thankfully, the firmware is not signed, let’s update the firmware!
- .. and try not to brick anything

Next

Let’s patch the firmware!

Firmware integrity

CRC?

```
ZALMAN_VE500_3637E_FWUpdater_V1.10/FW/INIC3637E_ISO_TOUCH_V110.bin
0001 FFE0: 25 C9 36 10 00 00 00 00  00 00 00 00 00 00 00 00  %.6..... ........
0001 FFF0: 00 00 00 00 FC BF 01 00  36 90 36 10 D0 B8 00 00  ........ 6.6.....
0002 0000:
ZALMAN_VE500_3637E_FWUpdater_V1.11/FW/INIC3637E_ISO_TOUCH_V111.bin
0001 FFE0: 25 C9 36 10 00 00 00 00  00 00 00 00 00 00 00 00  %.6..... ........
0001 FFF0: 00 00 00 00 FC BF 01 00  36 90 36 10 EF C9 00 00  ........ 6.6.....
```

Is that a CRC 16?

Use the DLL!

`iCommon.dll` exports `CInitioDevice::CalCRC(unsigned char *, int)` function.
We'll reuse this one!

Assembling patches

No really working assembler for ARCCompact

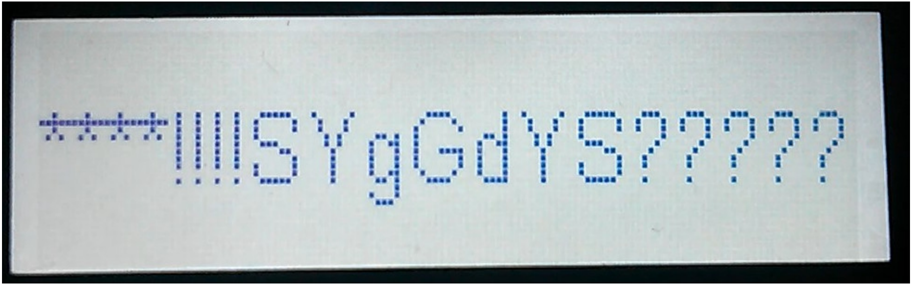
- Copy paste bytes
- Build small shellcodes

Example:

```
#Input genuine firmware
data = open("INIC3637E_ISO_TOUCH_V111.bin","rb").read()

body = data[:-4]
#apply patches on body

offset = 0x3838

(body, offset) = patch_data(body, offset, "08 75".replace(" ","").decode("hex")) #mov     r12, r0 ; copy keys buffer

(body, offset) = patch_data(body, offset, "CF 76 01 00 3C 0F".replace(" ","").decode("hex"))     #mov     r14, PIN
(body, offset) = patch_data(body, offset, "00 E5".replace(" ","").decode("hex"))                 #add     r13, r13, 0
(body, offset) = patch_data(body, offset, "0F D9".replace(" ","").decode("hex"))                 #mov     r1, 0xF
(body, offset) = patch_data(body, offset, "08 DC".replace(" ","").decode("hex"))                 #mov     r12, 8
...
```

Looking at memory

We were able to re-use the *Display string function* to print memory content on LCD:



Weird AES

Patching AES

AES was not “standard” so we:

- set the tweak to 0
- patched parameters to use ECB
- patched keys to compare to reference implementations

Result

Key is byteswapped and key 1 and key 2 are swapped.
Tweak is the sector’s LBA, in little endian.

Bruteforcer

Simple bruteforcer (OpenSSL/OpenMP): all possible PINs in 6s.

Firmware 2.0

New version: security fix?

- bruteforcer does not work anymore

Reverse new version

- PIN is now padded with `0xFD` instead of `0x00`

Consequences

- update bruteforcer
- probably a fix for PIN collisions

Encryption keys?

Decrypted secret block:

```
0000   20 49 4e 49 64 00 00 00 0f 2a 46 f6 00 00 00 00   INId....*F......
0010   20 49 4e 49 d8 6b 00 00 00 00 00 00 00 00 00 00   INI.k..........
[...]  almost only zeros
0100   45 3d 67 10 89 57 2d 70 88 cf 64 9f 8d 35 7e da  E=g..W-p..d..5~.
0110   e5 7b 33 24 c3 f3 94 23 15 2b fe f5 45 16 43 65  .{3$...#.+..E.Ce
0120   c7 de 10 0d 5d ef 30 fa 26 b8 e6 fe 5d 79 4e bd  ....].0.&...]yN.
0130   f5 a2 0b 2c 61 97 41 b6 01 3f 99 a4 67 45 a7 45  ...,a.A..?..gE.E
0140   32 db 89 8f be c2 43 81 95 46 6c 96 38 40 57 64  2.....C..Fl.8@Wd
0150   81 0a 93 1b 01 0b 9a 61 6e 28 54 50 71 51 f6 17  .......an(TPqQ..
[...]  high entropy
01d0   de ad 69 47 49 7e 75 87 de 0d 31 7a 80 d9 d2 af  ..iGI~u...1z....
01e0   03 7e 3d ff f2 63 39 11 b8 ef fd 15 6e 15 72 8c  .~=..c9.....n.r.
01f0   51 b2 ea 1c 1a 76 a7 79 ba 20 ea 18 f8 9c 3d 24  Q....v.y. ....=$
```

Probably the disk encryption keys.

To sum up

A few big issues

- disk keys stored on drive
- PIN is easily bruteforced
- one AES key is only zeros

One attack

- with drive only: recovering of PIN in 6s

Zalman drives summary

Table 1: summary of security properties

property	SHE500	VE500
basic crypto	OK	OK
disk tied to enclosure	OK	**NOT** OK
secrets stored securely	**NOT** OK	**NOT** OK
random drive key	**NOT** OK	OK (?)

Suppliers

Weird things

AES “pi” keys

Present in (see [AKm15]):

- JMicron chips (JMS538S): WD mainly
- Initio chips (1607E, 3607E): WD, Lenovo, Apricorn, Zalman,
- PLX chips (OXUF943SE): WD

Same AES modes constants

- Western digital drives (with JMicron)
- Initio code
- in Mac unlocker `WD Security.app` [WD] includes `.h` headers, created in 2006

Trying to find an explanation

Single IP?

Hypothesis:

- single Verilog/VHDL IP,
- with example code,
- and heavy copy paste by JMicron/Initio/PLX?

Consequences

- no actual diversity
- one vulnerability to rule them all?

A better design

A cheap, usable solution

Before all

Hire a cryptographer.

User-friendly: on disk secrets / master key

- easy support: data remains accessible if enclosure is broken
- no real security possible (512 bits to display?)
- only thing to do: “slow” hash + long (16) PIN

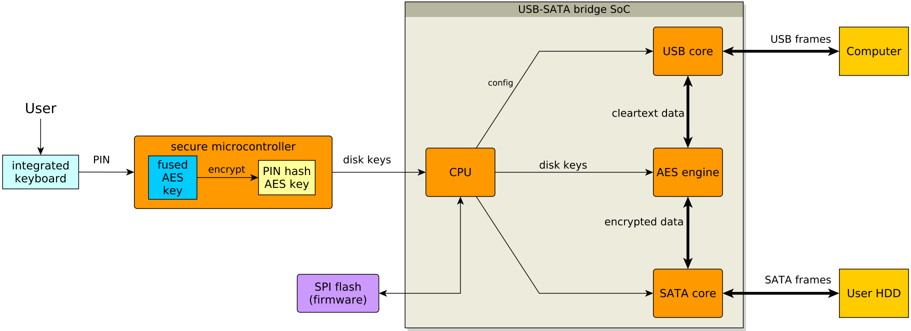
Less user-friendly: secrets in the enclosure

Make it harder for the attacker to access them:

- stored on a component that cannot be read programmatically

For example, using a PIC or AVR microcontroller (but dumpable for 1000-5000USD)

Best design



- use a secure component with a crypto engine, using a fuse programmable key
- provision the microcontroller with a random AES key (fuse blowing)
- encrypt the PIN's hash and disk keys with the AES engine

$\Longrightarrow$ **the attacker needs to physically attack each controller**

Conclusion

Conclusion

On the 2 drives

- two different companies but two failures: crypto design is hard.
- vulnerabilities reported in June, firmware updates followed.

What should manufacturers do

- hire cryptographers for the crypto design
- publish crypto design

Take away

- two disks broken in 1 man-month
- don’t trust products by default, audit them!
- don’t be scared, try, it’s fun :)

Thank you!

Thank you !

Questions?

See also our paper for more details.

References

- [AKm15] Gunnar Alendal, Christian Kison, and modg. got hw crypto? on the (in)security of a self-encrypting drive series. https://eprint.iacr.org/2015/1002.pdf, 2015.
- [CR15] Joffrey Czarny and Raphaël Rigo. Analysis of an encrypted hdd. SSTIC conference: article, 2015.
- [Dom10] Sprite (Jeroen Domburg). Sprite’s mods DiskGenie review. http://spritesmods.com/?art=diskgenie, 2010.
- [O’F16] Colin O’Flynn. Brute-forcing lockdown harddrive pin codes. https://www.blackhat.com/us-16/briefings.html#brute-forcing-lockdown-harddrive-pin-codes, 2016.
- [WD] WD Security for Mac: http://support.wdc.com/downloads.aspx?p=158&lang=en