I.MX MEMORY MADNESS

HOW TO DUMP, PARSE, AND ANALYZE I.MX FLASH MEMORY CHIPS

Damien Cauquil | HITB Amsterdam 2019 (🎂🎉)

digital.security

WHO AM I ?

- Head of R&D @ Econocom Digital.Security
- Senior security researcher
- Hardware hacker (or at least pretending)

AGENDA

- Firmware extraction 101
- Meet the i.MX architecture
- i.MX flash memory layout
- imx-nand-tools FTW
- Best practices

FIRMWARE EXTRACTION 101

WHY DO WE WANT TO EXTRACT A DEVICE'S FIRMWARE ?

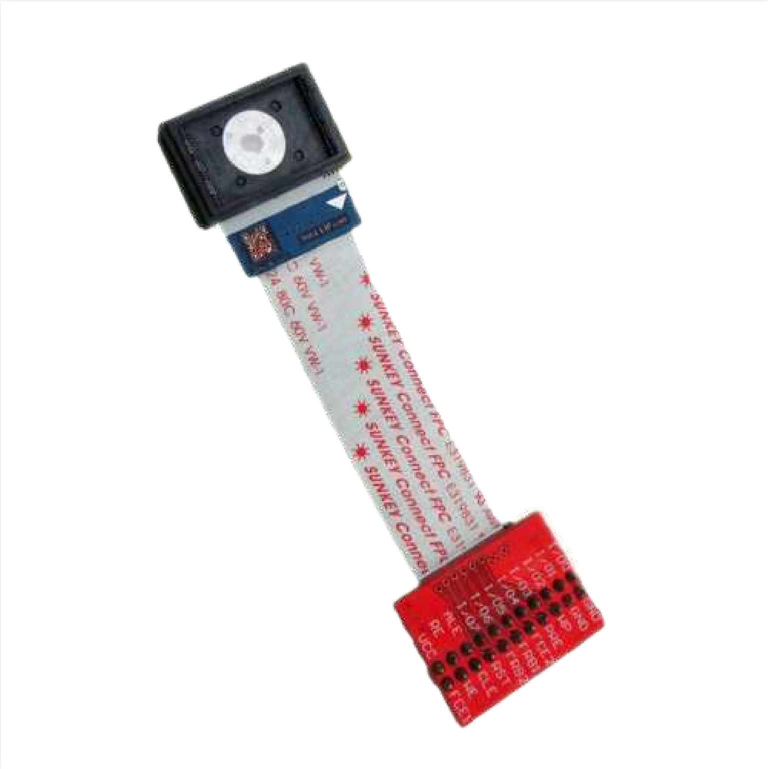
- Contains **filesystems, applications, binary files**
- May also contain **VERY interesting data**: encryption/decryption keys, certificates, passwords

WHY DO WE WANT TO EXTRACT A DEVICE'S FIRMWARE ?

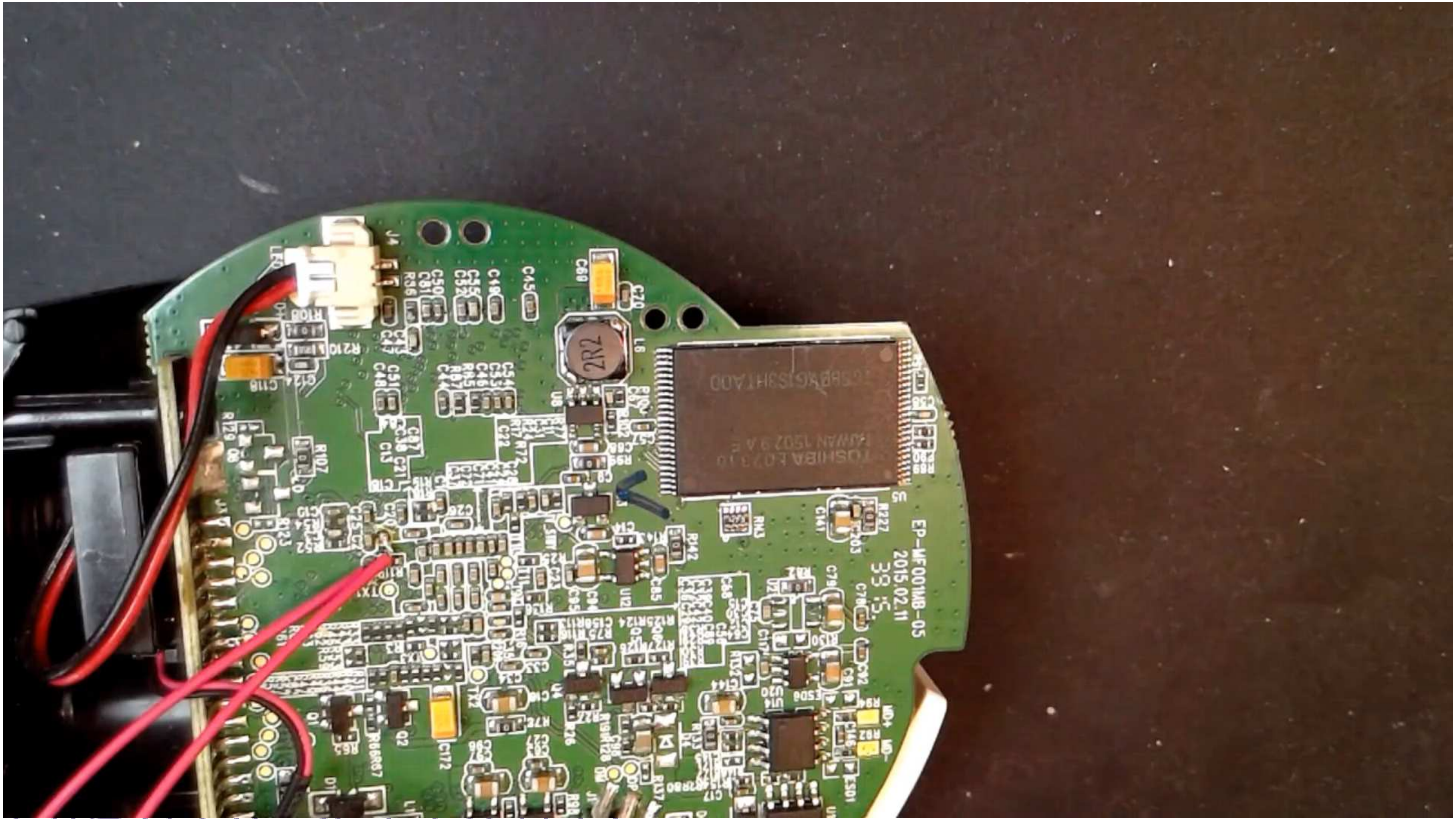
<u>We need to understand everything</u> about a device:

- How it has been designed
- How it (really) works
- Where and how every bit of data is stored

METHOD #1: CLIPPING & READING

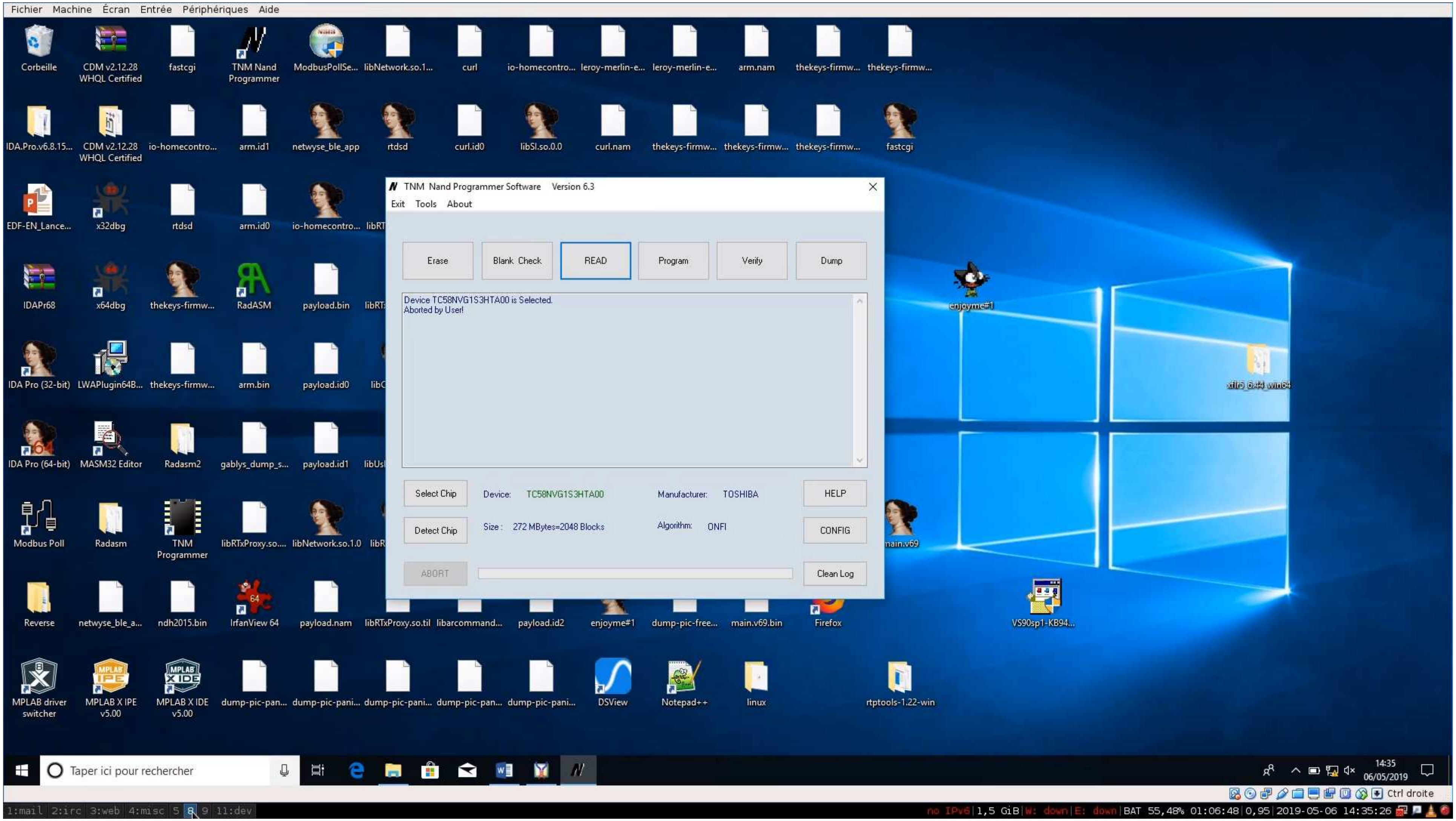


METHOD #2: CHIP-OFF



PROFESSIONAL FLASH PROGRAMMER

digital.security


TNM Nand Programmer Software Version 6.3
Exit Tools About
Erase
Blank Check
READ
Program
Verify
Dump
Device TC58NVG1S3HTA00 is Selected.
Aborted by User!
Select Chip
Device: TC58NVG1S3HTA00
Manufacturer: TOSHIBA
HELP
Detect Chip
Size : 272 MBytes=2048 Blocks
Algorithm: ONFI
CONFIG
ABORT
Clean Log
Taper ici pour rechercher
14:35
06/05/2019

NAND DUMP SIZE

```
$ ls -alh camera.bin
-rwx------ 1 virtualabs virtualabs 1,1G camera.bin
```

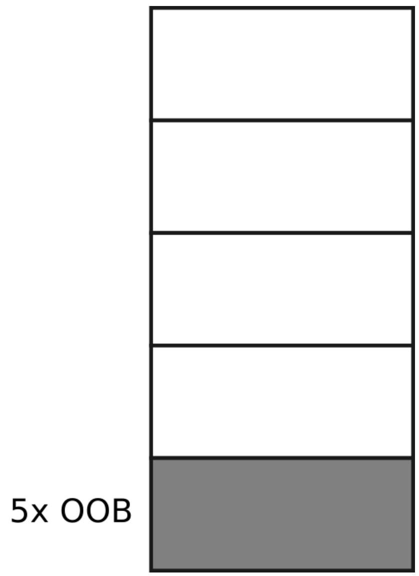
Dump file is greater than 1 GB !

PAGES, BYTES AND OOB

- Bytes are stored, erased, and modified in **pages**
- NAND flash chips **are not 100% reliable** and errors when storing bits may occur
- To avoid this, vendors usually provide **more space** to store Error Correction Codes (ECC) in spare-byte area (**OOB**)

digital.security

PAGES, BYTES AND OOB



digital.security

PAGES, BYTES AND OOB

MT29F16G08ADBCA , MT29F16G16ADBCA

Features

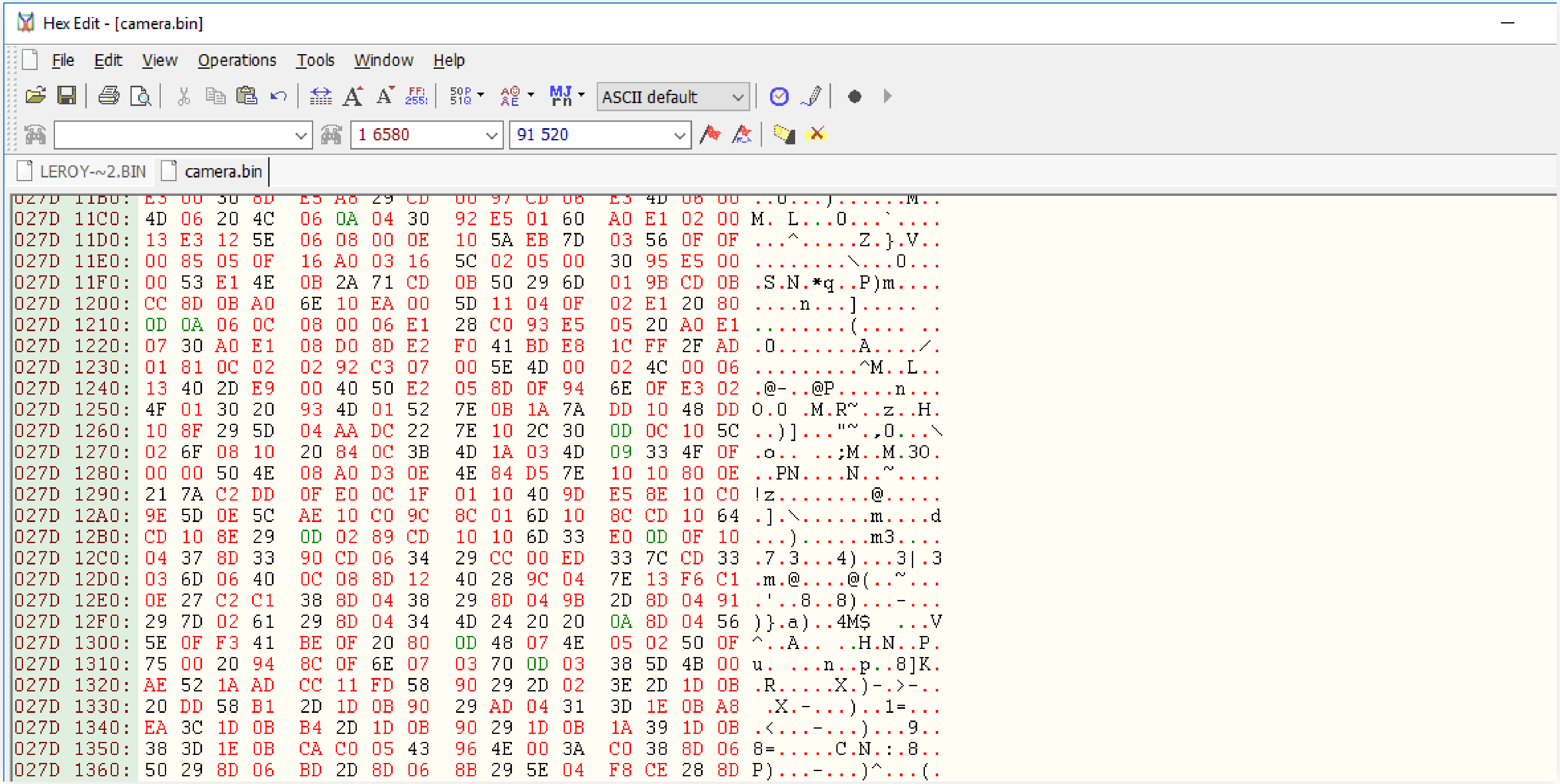
- Open NAND Flash Interface (ONFI) 1.0-compliant[1]
- Single-level cell (SLC) technology
- Organization
  - Page size x8: 4320 bytes (4096 + 224 bytes)
  - Page size x16: 2160 words (2048 + 112 words)
  - Block size: 64 pages (256K + 14K bytes)
  - Plane size: 2 planes x 2048 blocks per plane
  - Device size: 8Gb: 4096 blocks
  - Device size: 16Gb: 8192 blocks
- Asynchronous I/O performance
  - ^{t}RC/^{t}WC: 20ns (3.3V), 30ns (1.8V)

- First l
  ped fr
  ECC,
- RESE
  powe
- Altern
  powe
- Intern
  plane
- Quali
  - Da
    cat
  - En

REMOVING THE OOB DATA

```
import sys

PAGE, OOB = 4096, 224
BLOCK = PAGE + OOB
orig_dump = open(sys.argv[1], 'rb').read()
out_dump = open(sys.argv[2], 'wb')
nblocks = int(len(orig_dump) / BLOCK)
for i in range(nblocks):
        out_dump.write(orig_dump[i*BLOCK:(i+1)*PAGE + OOB])
out_dump.close()
orig_dump.close()
```



Hex Edit - [camera.bin]
File Edit View Operations Tools Window Help
ASCII default
1 6580
91 520
LEROY-~2.BIN
camera.bin
027D 11C0: 4D 06 20 4C 06 0A 04 30 92 E5 01 60 A0 E1 02 00 M. L...0...`....
027D 11D0: 13 E3 12 5E 06 08 00 0E 10 5A EB 7D 03 56 0F 0F ...^.....Z.}.V..
027D 11E0: 00 85 05 0F 16 A0 03 16 5C 02 05 00 30 95 E5 00 ....\...0...
027D 11F0: 00 53 E1 4E 0B 2A 71 CD 0B 50 29 6D 01 9B CD 0B .S.N.*q..P)m....
027D 1200: CC 8D 0B A0 6E 10 EA 00 5D 11 04 0F 02 E1 20 80 ....n...]..... .
027D 1210: 0D 0A 06 0C 08 00 06 E1 28 C0 93 E5 05 20 A0 E1 ....(.... ..
027D 1220: 07 30 A0 E1 08 D0 8D E2 F0 41 BD E8 1C FF 2F AD .0.......A..../.
027D 1230: 01 81 0C 02 02 92 C3 07 00 5E 4D 00 02 4C 00 06 ....^M..L..
027D 1240: 13 40 2D E9 00 40 50 E2 05 8D 0F 94 6E 0F E3 02 .@-..@P.....n...
027D 1250: 4F 01 30 20 93 4D 01 52 7E 0B 1A 7A DD 10 48 DD O.0 .M.R~..z..H.
027D 1260: 10 8F 29 5D 04 AA DC 22 7E 10 2C 30 0D 0C 10 5C ..)]..."~.,0...\
027D 1270: 02 6F 08 10 20 84 0C 3B 4D 1A 03 4D 09 33 4F 0F .o.. ..;M..M.3O.
027D 1280: 00 00 50 4E 08 A0 D3 0E 4E 84 D5 7E 10 10 80 0E ..PN....N..~....
027D 1290: 21 7A C2 DD 0F E0 0C 1F 01 10 40 9D E5 8E 10 C0 !z........@.....
027D 12A0: 9E 5D 0E 5C AE 10 C0 9C 8C 01 6D 10 8C CD 10 64 .].\......m....d
027D 12B0: CD 10 8E 29 0D 02 89 CD 10 10 6D 33 E0 0D 0F 10 ...).......m3....
027D 12C0: 04 37 8D 33 90 CD 06 34 29 CC 00 ED 33 7C CD 33 .7.3...4)...3|.3
027D 12D0: 03 6D 06 40 0C 08 8D 12 40 28 9C 04 7E 13 F6 C1 .m.@....@(..~...
027D 12E0: 0E 27 C2 C1 38 8D 04 38 29 8D 04 9B 2D 8D 04 91 .'..8..8)...-...
027D 12F0: 29 7D 02 61 29 8D 04 34 4D 24 20 20 0A 8D 04 56)}.a)..4M$ ...V
027D 1300: 5E 0F F3 41 BE 0F 20 80 0D 48 07 4E 05 02 50 0F ^..A.. ..H.N..P.
027D 1310: 75 00 20 94 8C 0F 6E 07 03 70 0D 03 38 5D 4B 00 u. ...n..p..8]K.
027D 1320: AE 52 1A AD CC 11 FD 58 90 29 2D 02 3E 2D 1D 0B .R.....X.)-.>-..
027D 1330: 20 DD 58 B1 2D 1D 0B 90 29 AD 04 31 3D 1E 0B A8 .X.-...)..1=...
027D 1340: EA 3C 1D 0B B4 2D 1D 0B 90 29 1D 0B 1A 39 1D 0B .<...-...)...9..
027D 1350: 38 3D 1E 0B CA C0 05 43 96 4E 00 3A C0 38 8D 06 8=.....C.N.:.8..
027D 1360: 50 29 8D 06 BD 2D 8D 06 8B 29 5E 04 F8 CE 28 8D P)...-...)^...(.

CHECKING OUR DUMP WITH BINWALK

```
$ binwalk ipcam.fw.bin

DECIMAL         HEXADECIMAL     DESCRIPTION
-----------------------------------------------------------
96188           0x177BC         CRC32 polynomial table, [...]
[...]
2490368         0x260000        Squashfs filesystem, [...]
4456448         0x440000        Squashfs filesystem, [...]
5505024         0x540000        Squashfs filesystem, [...]
6684672         0x660000        Squashfs filesystem, [...]
7208960         0x6E0000        JFFS2 filesystem, little endian
7643512         0x74A178        JFFS2 filesystem, little endian
```

digital.security

EXTRACTING FILES FROM VARIOUS FILESYSTEMS

- SquashFS: compressed filesystem, one partition/image
- YAFFS2: Yet Another Flash FS
- JFFS2: Journalized Flash FS version 2, one partition/image
- UBI: Unsorted Block Image, multiple partitions with various FS

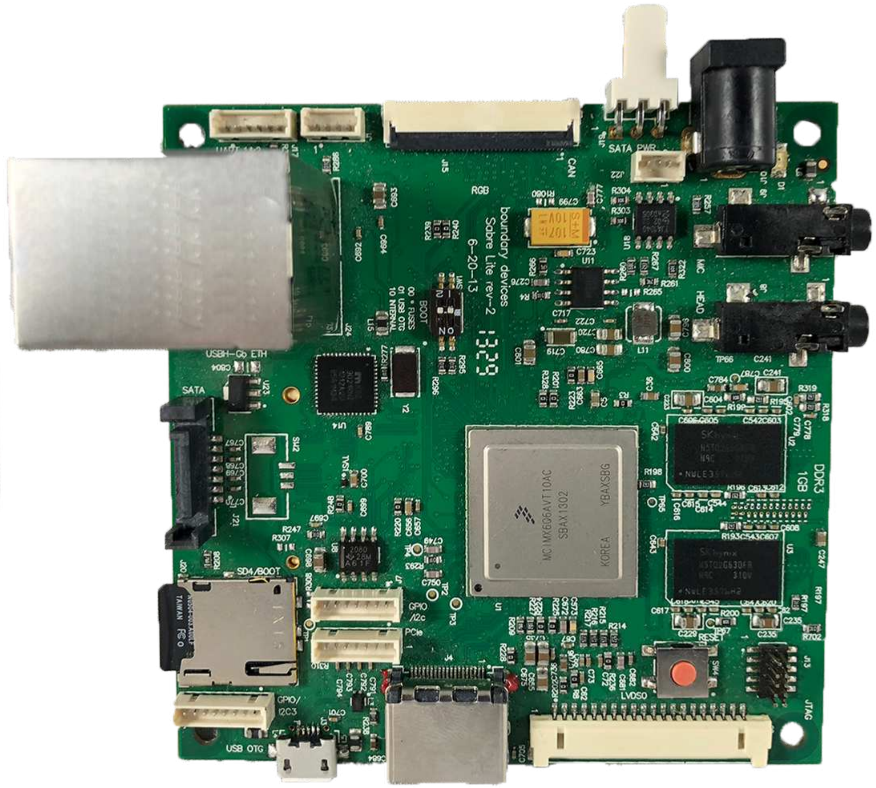
digital.security

IT'S A DOCUMENTED PROCESS

PenTestPartners just published a blog entry:

http://bit.ly/HITB-PTPFW

AND WE STUMBLED UPON AN I.MX6 SYSTEM



digital.security

```
virtualabs@virtubox:~$ 
```

digital.security


4GIFs.com

HEX ANALYSIS REVEALED WEIRD BYTES

```
3D81:E2E0 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
3D81:E2F0 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
3D81:E300 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
3D81:E310 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
3D81:E320 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
3D81:E330 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
3D81:E340 FF 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ÿ...............
3D81:E350 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
3D81:E360 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
3D81:E370 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
3D81:E380 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
3D81:E390 00 00 00 00  00 00 00 00  00 00    00  00 00 00 00  ................
3D81:E3A0 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
3D81:E3B0 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
3D81:E3C0 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
3D81:E3D0 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
3D81:E3E0 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
```

A CRAPPY BYTE BEFORE UBI SIGNATURE

```
30C3:4790 FF FF FF FF  FF FF FF FF  FF FF FF FF  FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
30C3:47A0 FF FF FF FF  FF FF FF FF  FF FF FF FF  FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
30C3:47B0 FF FF FF FF  FF FF FF FF  FF FF FF FF  FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
30C3:47C0 FF FF FF FF  FF FF FF FF  FF FF FF FF  FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
30C3:47D0 FF FF FF FF  FF FF FF FF  FF FF FF FF  FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
30C3:47E0 FF FF FF FF  FF FF FF FF  FF FF FF FF  FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
30C3:47F0 FF FF FF FF  FF FF FF FF  FF FF FF FF  FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
30C3:4800 00 55 42 49  23 01 00 00  00 00 00 00  00 00 00 00  .UBI#...........
30C3:4810 02 00 00 10  00 00 00 20  00 07 42 E4  66 00 00 00  ........  ..Bäf...
30C3:4820 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
30C3:4830 00 00 00 00  00 00 00 00  00 00 00 00  00 15 87 C5  ...............Å
30C3:4840 78 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  x...............
30C3:4850 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
30C3:4860 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
```

SAME 1-BYTE OFFSET IN BINWALK OUTPUT

```
24891865      0x17BD1D9      YAFFS filesystem
25159701      0x17FE815      YAFFS filesystem
25436181      0x1842015      YAFFS filesystem
25712661      0x1885815      YAFFS filesystem
25727234      0x1889102      Unix path: /usr/share/brw/local/index.html  0
35389441      0x21C0001      UBI erase count header, version: 1, EC: 0x4, VID header offset: 0x1000,
```

UBI header is not aligned on page size (0x1000)

THAT'S WEIRD 😕

- Quick investigation revealed **anomalies**
- Our **dump seems OK**, but we still cannot extract data from it
- It must be related to **i.MX**: maybe a custom storage mechanism

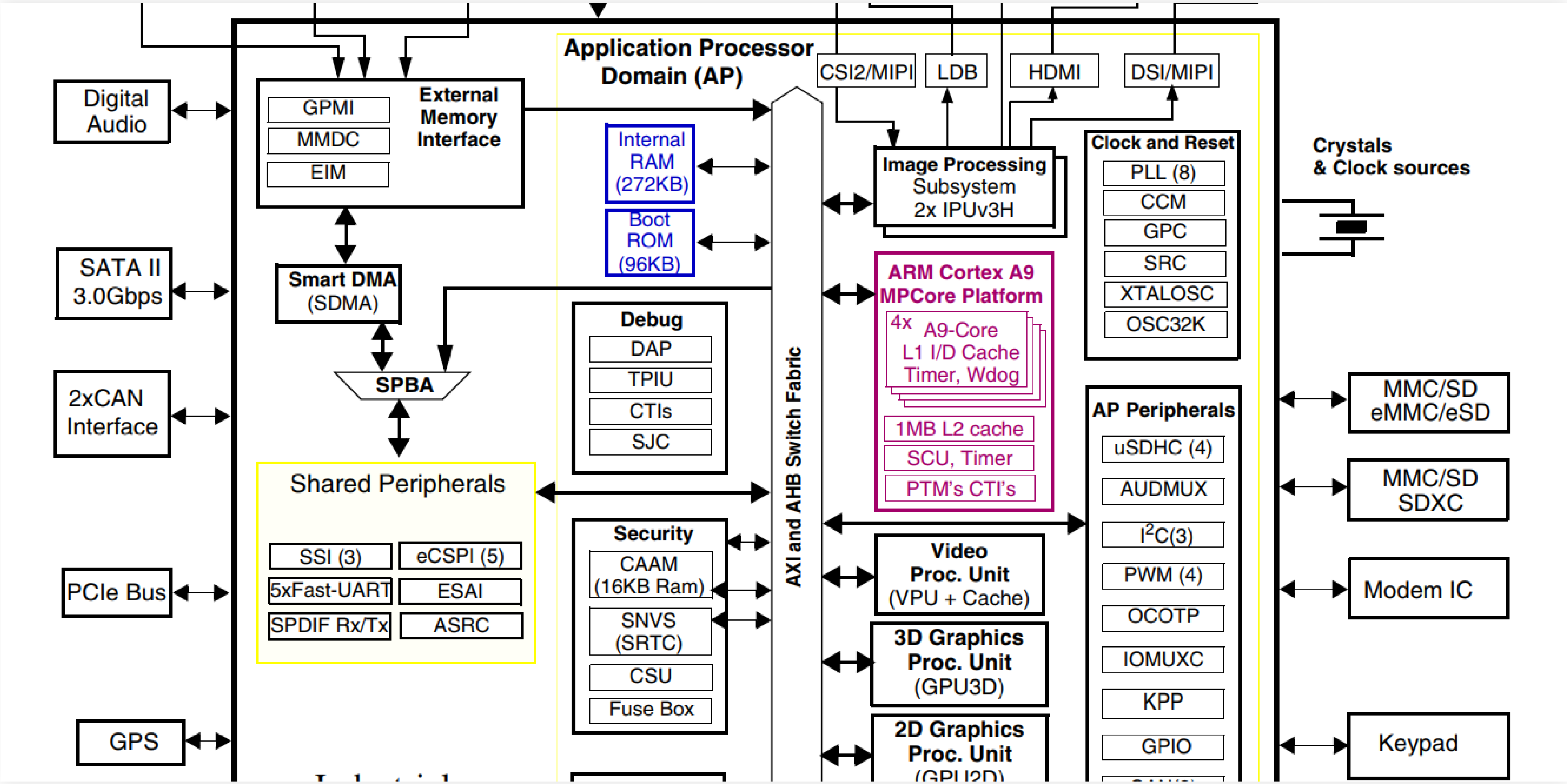
digital.security

I.MX ARCHITECTURE AND MEMORY LAYOUT

I.MX ARCHITECTURE

- Integrated Multimedia Application processors
- Popular in **automotive and home automation** industries
- Provides a lot of features including:
    - Secure/non-secure RAM
    - SATA II support
    - Secure Boot ...

I.MX ARCHITECTURE



digital.security

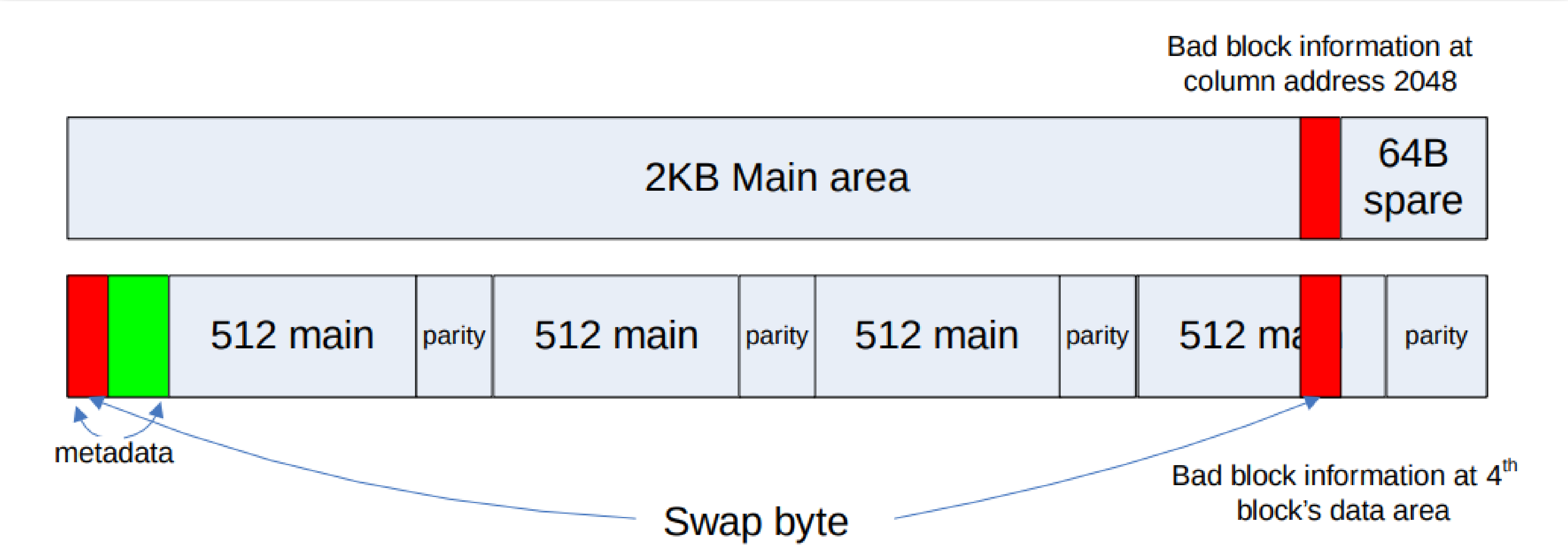
I.MX ARCHITECTURE

- Can boot on various storage devices:
    - NAND Flash
    - Parallel NOR Flash
    - SD card
    - MMC
    - SATA HDD
- It also embeds a boot ROM (Freescale Inc.)

GENERAL-PURPOSE MULTIMEDIA INTERFACE

- **controls how data is read/stored** on NAND flash chips
- supports **multiple NAND flash chips**
- uses **BCH** to perform error control and correction

NAND FLASH STRUCTURE



(image extracted from i.MX28 reference manual)

HOW IS DATA STORED ?

- Data is split in **512-byte chunks**
- **ECC bits** are added at the end of each chunk
- Chunks are then **grouped and stored in a page** preceeded by one metadata block
- Bad block **marker byte is swapped** with first metadata byte !

WEIRD BYTE EXPLAINED !

```
3D81:E2E0 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
3D81:E2F0 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
3D81:E300 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
3D81:E310 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
3D81:E320 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
3D81:E330 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
3D81:E340 FF 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ÿ...............
3D81:E350 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
3D81:E360 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
3D81:E370 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
3D81:E380 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
3D81:E390 00 00 00 00  00 00 00 00  00 00    00  00 00 00 00  ................
3D81:E3A0 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
3D81:E3B0 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
3D81:E3C0 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
3D81:E3D0 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
3D81:E3E0 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
```

digital.security

FIRMWARE CONFIGURATION BLOCK (FCB)

- This structure contains **all the required information** about how data is stored
- It must be present in the **first 1MB**
- Second field of this structure contains **"FCB "** in ASCII

FCB SIGNATURE IN HEXDUMP

```
0000:0000 00 00 00 00  00 00 00 00  00 00 00 00  0D FB FF FF  .............ûÿÿ
0000:0010 46 43 42 20  00 00 00 01  50 3C 19 06  00 00 00 00  FCB ....P<......
0000:0020 00 10 00 00  E0 10 00 00  40 00 00 00  00 00 00 00  ....à...@.......
0000:0030 00 00 00 00  00 00 00 00  08 00 00 00  00 02 00 00  ................
0000:0040 00 02 00 00  08 00 00 00  01 00 00 00  07 00 00 00  ................
0000:0050 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
0000:0060 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
0000:0070 00 00 00 00  C2 15 00 00  42 11 00 00  36 00 00 00  ....Â...B...6...
0000:0080 36 00 00 00  00 01 00 00  49 0F 00 00  00 00 00 00  6.......I.......
0000:0090 00 10 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
0000:00A0 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
0000:00B0 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
0000:00C0 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
0000:00D0 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
0000:00E0 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
0000:00F0 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
0000:0100 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
```

FIRMWARE CONFIGURATION BLOCK (FCB)

- NAND page data size
- Block N ECC type
- Block N size
- Block 0 ECC type
- Block 0 size
- Number of bytes in metadata of a page
- ...

FCB SIGNATURE IN HEXDUMP

```
0000:0000 00 00 00 00 00 00 00 00 00 00 00 00 0D FB FF FF .............ûÿÿ
0000:0010 46 43 42 20 00 00 00 01 50 3C 19 06 00 00 00 00 FCB ....P<......
0000:0020 00 10 00 00 E0 10 00 00 40 00 00 00 00 00 00 00 ....à...@.......
0000:0030 00 00 00 00 00 00 00 00 08 00 00 00 00 02 00 00 ................
0000:0040 00 02 00 00 08 00 00 00 01 00 00 00 07 00 00 00 ................
0000:0050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0000:0060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0000:0070 00 00 00 00 C2 15 00 00 42 11 00 00 36 00 00 00 ....Â...B...6...
0000:0080 36 00 00 00 00 01 00 00 49 0F 00 00 00 00 00 00 6.......I.......
0000:0090 00 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0000:00A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0000:00B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0000:00C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0000:00D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0000:00E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0000:00F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0000:0100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

Offset **+0x3C:** number of bytes of metadata block

1-BYTE OFFSET EXPLAINED !

```
30C3:4790 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
30C3:47A0 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
30C3:47B0 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
30C3:47C0 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
30C3:47D0 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
30C3:47E0 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
30C3:47F0 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
30C3:4800 00554249 23010000 00000000 00000000 .UBI#...........
30C3:4810 02000010 00000020 000742E4 66000000 ....... ..Bäf...
30C3:4820 00000000 00000000 00000000 00000000 ................
30C3:4830 00000000 00000000 00000000 001587C5 ...............Å
30C3:4840 78000000 00000000 00000000 00000000 x...............
30C3:4850 00000000 00000000 00000000 00000000 ................
30C3:4860 00000000 00000000 00000000 00000000 ................
```

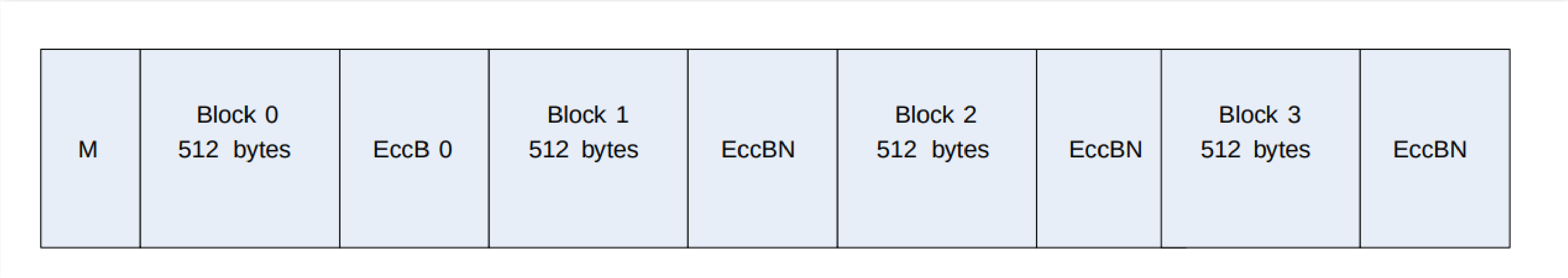
digital.security

DISCOVERED BAD BLOCK TABLE (DBBT)

- Provides **custom NAND bad block management**
- Its headers provide information about the **number of bad blocks and impacted pages**

```
0010:E000 00 00 00 00  00 44 42 42  54 00 00 00  01 00 00 00  .....DBBT.......
0010:E010 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
0010:E020 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
0010:E030 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
0010:E040 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
0010:E050 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
0010:E060 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
0010:E070 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
0010:E080 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
```

ECC



(image extracted from i.MX28 reference manual)

ECC

- Provides a way to dynamically fix errors, if possible
- Uses **BCH** (Bose, Ray-Chaudhuri and Hocquenghem) error-correcting code
- Data bytes **may be shifted by a number of bits** due to BCH bits

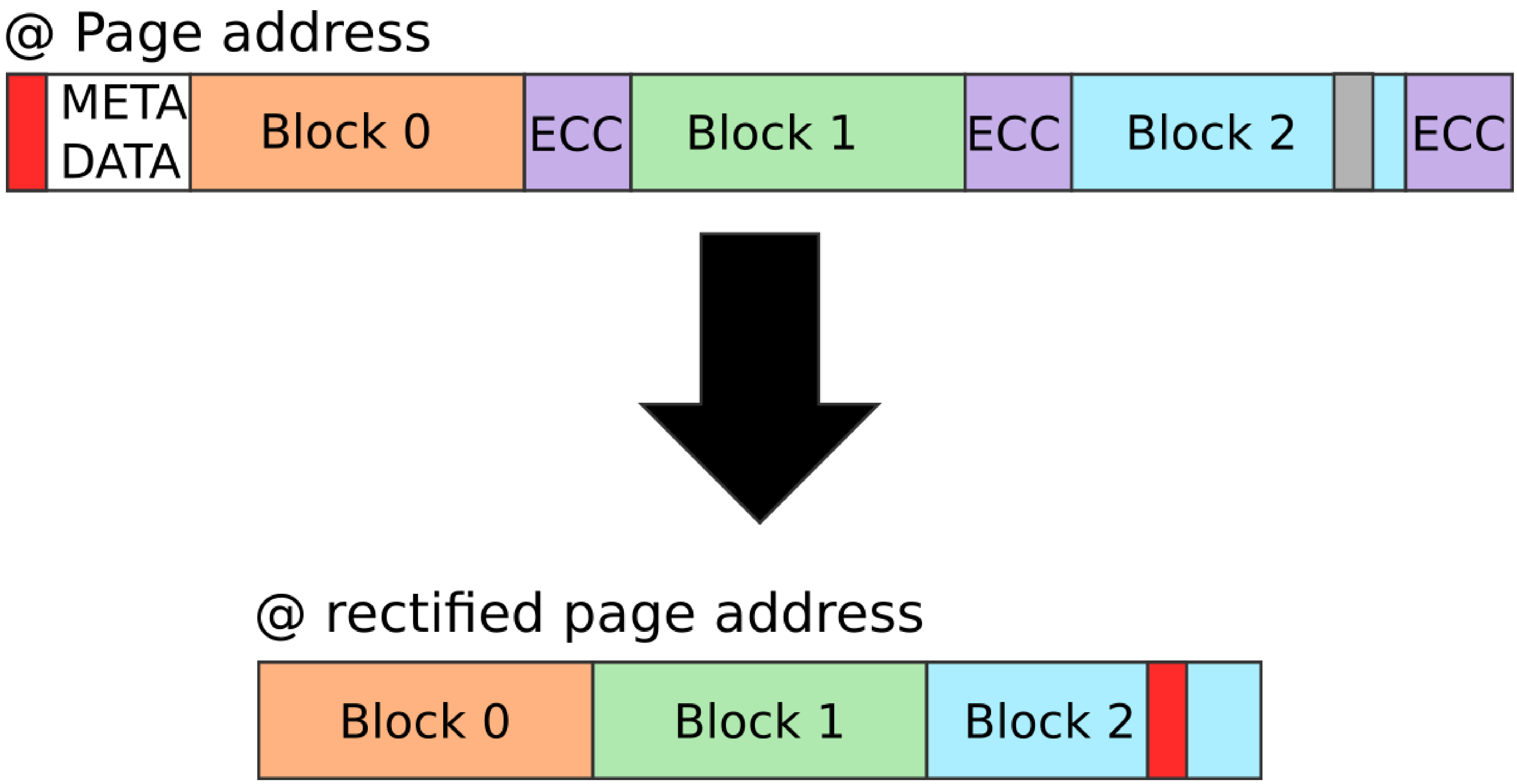
SO, WHAT'S NEXT ?

FROM NAND FLASH DUMP TO FILESYSTEMS

RECOVER AND REMAP ALL THE BYTES

- We first find an **FCB structure** and parse it to recover all the critical parameters
- Then we **remove every metadata and ECC bits** according to this FCB
- We use ECC bits to **fix errors and save each block** in an output file

digital.security



digital.security

IMX NAND TOOLS

```
$ sudo pip install imx-nand-tools
```

https://github.com/DigitalSecurity/imx-nand-tools/

digital.security

FCB PARSING

```
virtualabs@virtubox:~$
```

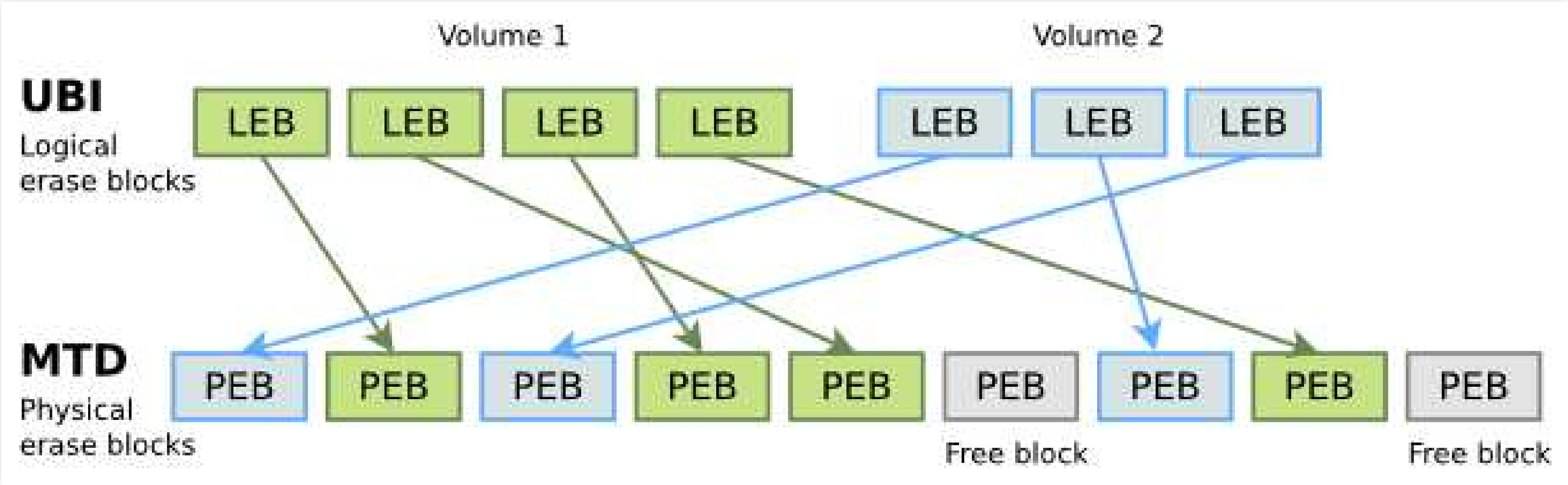
CONVERTING IMAGE TO USEABLE DUMP

```
virtualabs@virtubox:~$
```

ANALYZING THIS NEW DUMP

```
virtualabs@virtubox:~$
```

UBI OVERVIEW



digital.security

UBIREADER

- Provides a **set of tools** to parse, analyze and extract volumes and files from a UBI container
- **Open-source** and available on Github
- Written in **Python**
- Does not support *fastboot* mode

ACCESSING FILES STORED IN VARIOUS IMAGES

```
$ ubireader_extract_files -iw img-xx_vol-iio_0633_0.ubifs
[...]
$ ls ubifs-root/ -al
total 76
drwxr-xr-x 19 virtualabs virtualabs 4096 mai    9 09:40 .
drwxr-xr-x  3 virtualabs virtualabs 4096 mai    9 09:40 ..
drwxr-xr-x  2 virtualabs virtualabs 4096 mai    9 09:40 bin
drwxr-xr-x  2 virtualabs virtualabs 4096 mai    9 09:40 boot
drwxr-xr-x  5 virtualabs virtualabs 4096 mai    9 09:40 Data
[...]
drwxr-xr-x  2 virtualabs virtualabs 4096 mai    9 09:40 tmp
drwxr-xr-x  7 virtualabs virtualabs 4096 mai    9 09:40 usr
drwxr-xr-x  2 virtualabs virtualabs 4096 mai    9 09:40 var
```

THAT'S A WIN

SECURITY THROUGH OBSCURITY

(Image: XKCD #257)

digital.security

NOT SO OBSCURE AFTERALL

- **Reference manuals** describe how i.MX GPMI works and how data is read/stored on NAND flash memory
- **Publicly available code on Github** provides a better understanding of critical structures and how things are implemented

digital.security

IMX KNOBS GITHUB REPOSITORY

```
struct fcb_block {
        FCB_ROM_NAND_Timing_t  m_NANDTiming;             //!< Optimum timing parameters for Tas, Tds, Tdh in nsec.
        uint32_t        m_u32PageDataSize;              //!< 2048 for 2K pages, 4096 for 4K pages.
        uint32_t        m_u32TotalPageSize;             //!< 2112 for 2K pages, 4314 for 4K pages.
        uint32_t        m_u32SectorsPerBlock;           //!< Number of 2K sections per block.
        uint32_t        m_u32NumberOfNANDs;             //!< Total Number of NANDs - not used by ROM.
        uint32_t        m_u32TotalInternalDie;          //!< Number of separate chips in this NAND.
        uint32_t        m_u32CellType;                  //!< MLC or SLC.
        uint32_t        m_u32EccBlockNEccType;          //!< Type of ECC, can be one of BCH-0-20
        uint32_t        m_u32EccBlock0Size;             //!< Number of bytes for Block0 - BCH
        uint32_t        m_u32EccBlockNSize;             //!< Block size in bytes for all blocks other than Block0 - BCH
        uint32_t        m_u32EccBlock0EccType;          //!< Ecc level for Block 0 - BCH
```

digital.security

IMX UBOOT GITHUB REPOSITORY

```
static inline uint32_t mx28_nand_get_ecc_strength(uint32_t page_data_size,
                                                  uint32_t page_oob_size)
{
        int ecc_strength;

        /*
         * Determine the ECC layout with the formula:
         *      ECC bits per chunk = (total page spare data bits) /
         *              (bits per ECC level) / (chunks per page)
         * where:
         *      total page spare data bits =
         *              (page oob size - meta data size) * (bits per byte)
         */
        ecc_strength = ((page_oob_size - MXS_NAND_METADATA_SIZE) * 8)
                        / (MXS_NAND_BITS_PER_ECC_LEVEL *
                                mx28_nand_ecc_chunk_cnt(page_data_size));
```

Y U NO ENCRYPT ?

- i.MX systems support **NAND flash encryption**
- Most of the systems we have tested so far **do not use encryption (what did you expect ?)**

KNOWN VARIANTS

- Some i.MX dumps we made seemed to use a different ECC mechanism
- Various GPMI drivers mention **different versions** of Freescale ROM and variants of FCB structure
- The current version of *imx-nand-tools* worked for all of our dumps but may fail with yours, so ...

INSTALL, TEST, AND CONTRIBUTE !

CONCLUSION

- i.MX system uses a **custom NAND flash layout**
- This **layout is documented** in various documents and publicly available code
- **imx-nand-tools** provides a set of tools to handle this layout and convert dumps into useable images
- i.MX systems **should use NAND flash encryption feature** to avoid key/password/IP leaks

THANKS FOR ATTENDING, ANY QUESTION ?

Contact

damien.cauquil@digital.security

@virtualabs

RELATED LINKS

- **PTP firmware extraction tips & tricks:** https://www.pentestpartners.com/security-blog/how firmware-analysis-tools-tips-and-tricks/
- **IMX28 Reference manual:** https://bootlin.com/~maxime/pub/datasheet/MCIMX2
- **UBOOT NAND utility:** https://github.com/u-boot/u-boot/blob/master/tools/mxsboot.c
- **Freescale Linux driver:** https://github.com/Freescale/ fslc/tree/4.1-2.0.x-imx/drivers/mtd/nand/gpmi-nand