Dmitry Sklyarov

DECEMBER 4-7, 2017
EXCEL / LONDON, UK

Intel ME: Flash File System Explained

POSITIVE TECHNOLOGIES

#BHEU / @BLACK HAT EVENTS

Research Team

Maxim Goryachy

Mark Ermolov

Dmitry Sklyarov

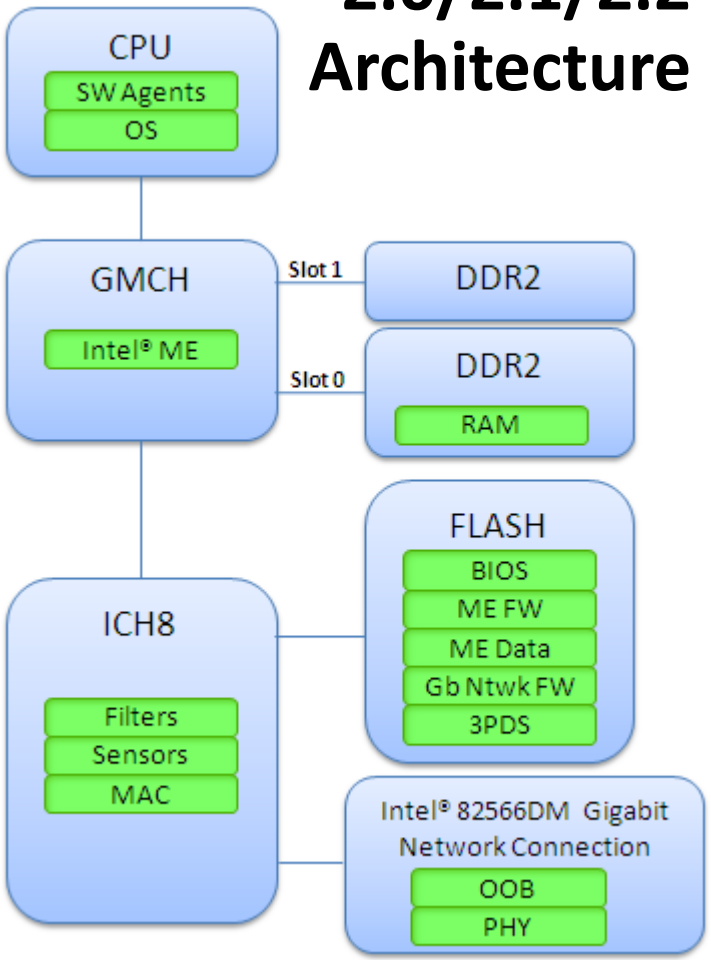
Outline

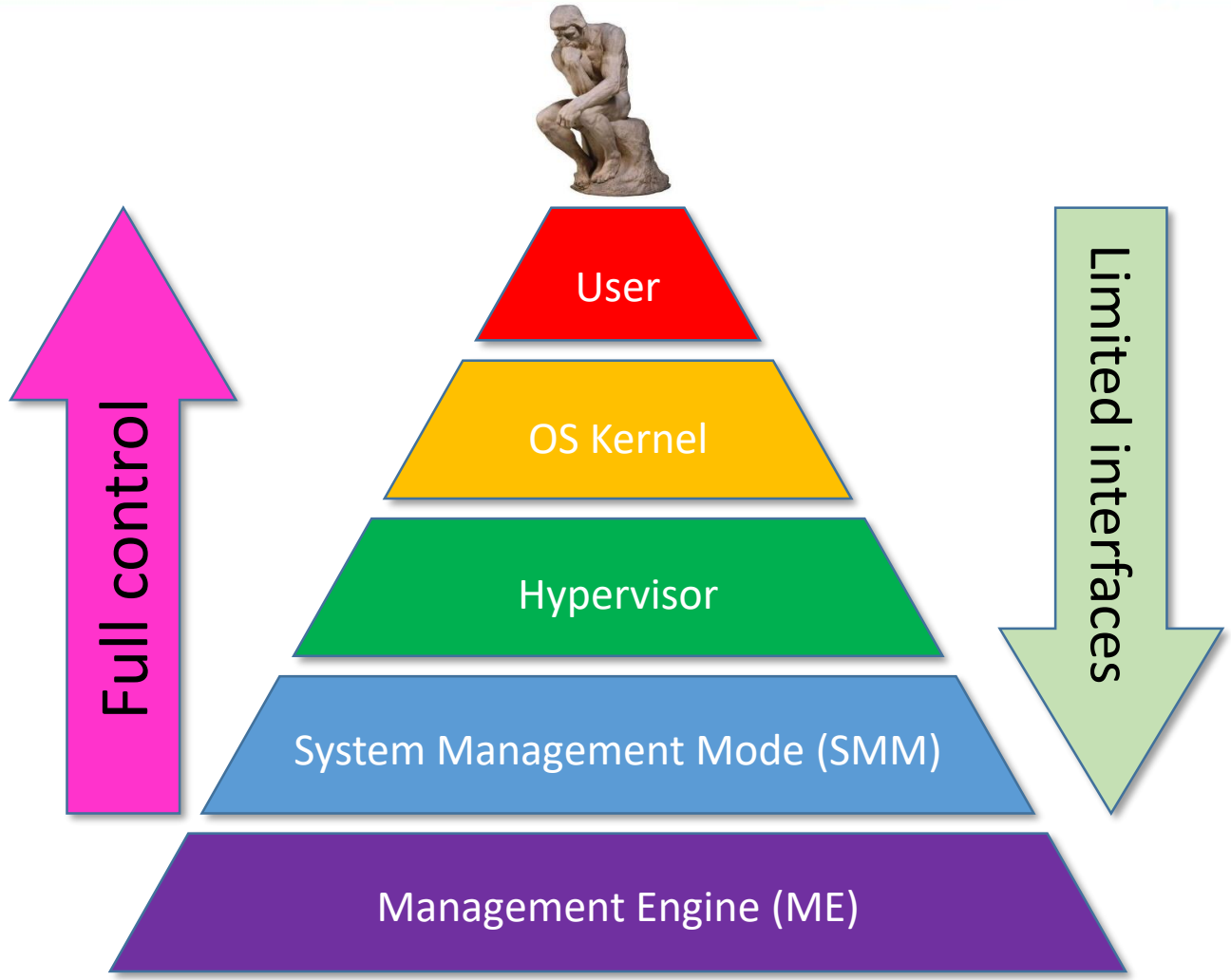
- Introduction
  - What is Intel ME
  - Notes about Flash File System design
- MFS Internals
  - MFS partition structure
  - File extraction
- MFS Usage
  - Special files
  - Integrity, Encryption, Anti-Replay
- Additional Info
  - VFS implementation in ME 11.x

Introduction

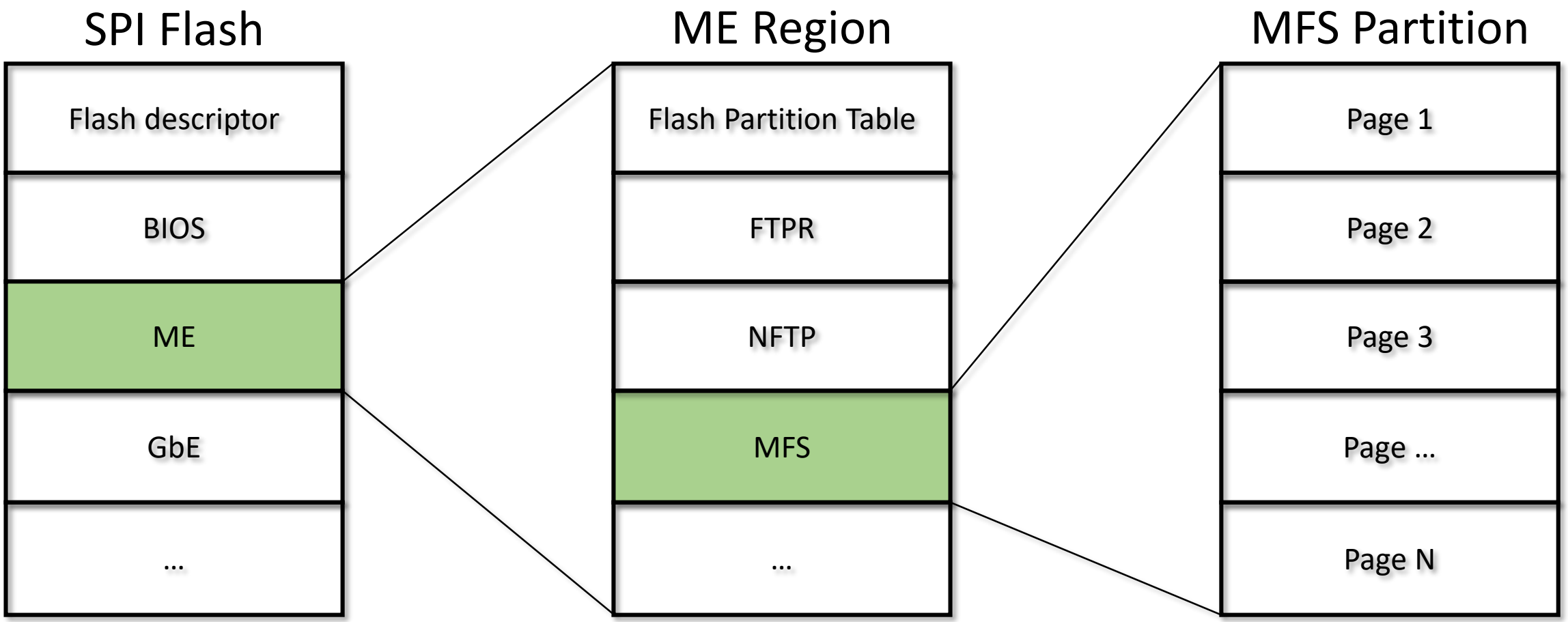
ME Position in Computer System

Intel AMT Release 2.0/2.1/2.2 Architecture





MFS Partition Layout



Flash Memory Characteristics

- Any byte can be written independently
- Need to erase (make all bits=1) before re-writing
- Erasing with precision of block (e.g., 8K) only
- Limited number of guaranteed erase cycles
  - Usually between 10,000 and 1,000,000
  - Inerasable block should be marked as “bad”

Flash File System Design Goals

- Erase count minimization
  Use incremental modification to avoid redundant erases

- Wear leveling
  Distribute erases between blocks as evenly as possible

Popular Linux Flash File Systems:

- JFFS, JFFS2, and YAFFS
- UBIFS
- LogFS
- F2FS

Recommended Materials

intel

Patents / White Papers / Documentation

Intel ME Secrets
Hidden code in your chipset and how to discover what exactly it does

Igor Skochinsky
Hex-Rays

RECON 2014
Montreal

ptresearch / unME11

MFS Internals

MFS Pagination

MFS is set of fixed-size pages (8192 == 0x2000 bytes each)

System pages

1/12 of total number of pages

Empty page

the only page without signature

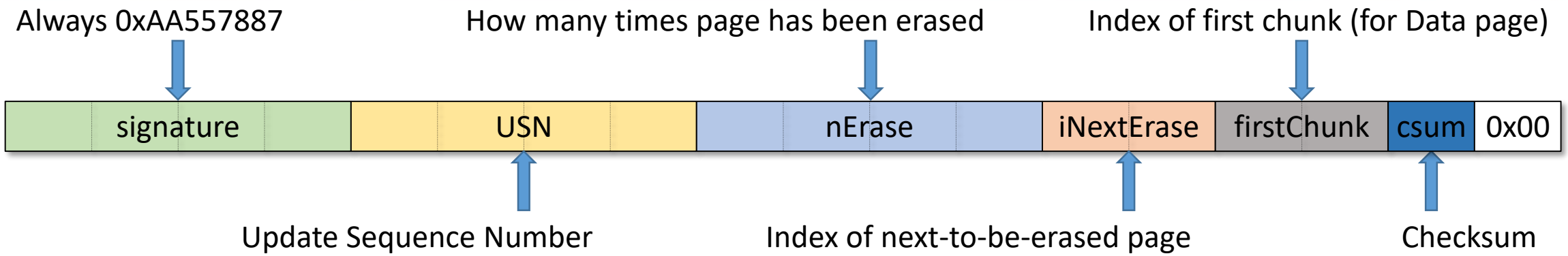
Data pages

all other pages

```
#define MFS_PAGE_SIZE 0x2000
cbMFS = sizeof(MFS); // Size of MFS partition
nPages = cbMFS / MFS_PAGE_SIZE; // Total number of pages

nSysPages = nPages / 12; // Number of System pages
nDataPages = nPages - nSysPages - 1; // Number of Data pages
```

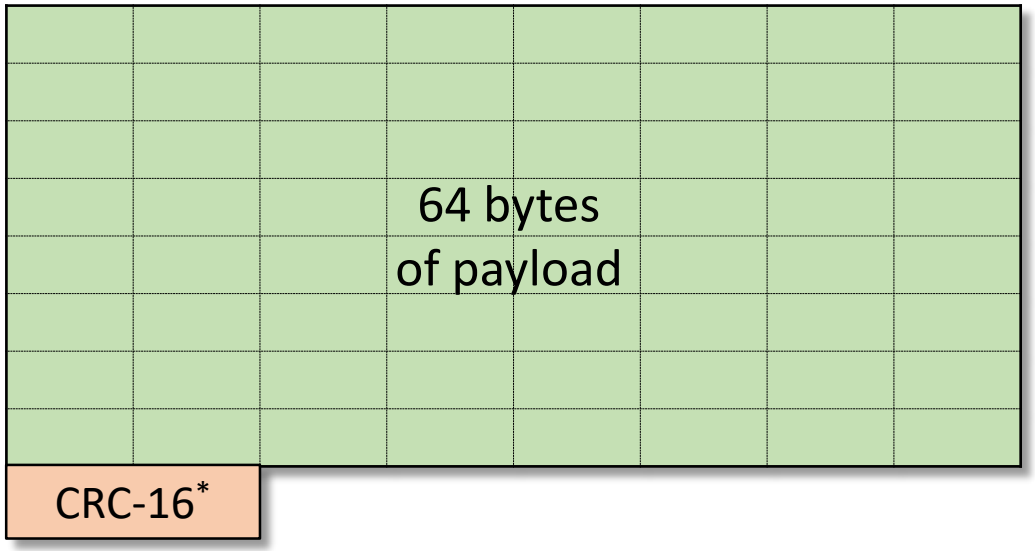
MFS Page Header

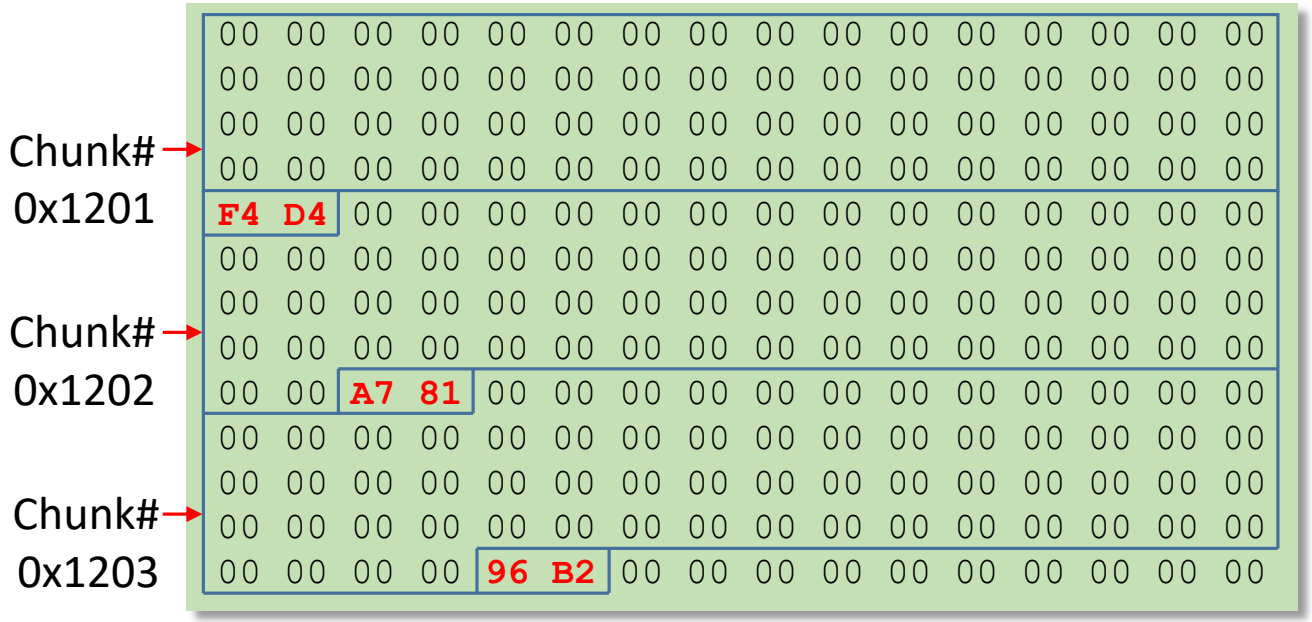



```
typedef struct {
  unsigned __int32 signature; // Page signature == 0xAA557887
  unsigned __int32 USN; // Update Sequence Number
  unsigned __int32 nErase; // How many times page has been erased
  unsigned __int16 iNextErase; // Index of next-to-be-erased page
  unsigned __int16 firstChunk; // Index of first chunk (for Data page)
  unsigned __int8 csum; // Page Header checksum (for first 16 bytes)
  unsigned __int8 b0; // Always 0
} T_MFS_Page_Hdr; // 18 bytes
```

Page Chunks

Single Chunk (66 bytes)





*CCITT CRC-16 calculated from
Chunk data + 16-bit (2-byte) Chunk Index

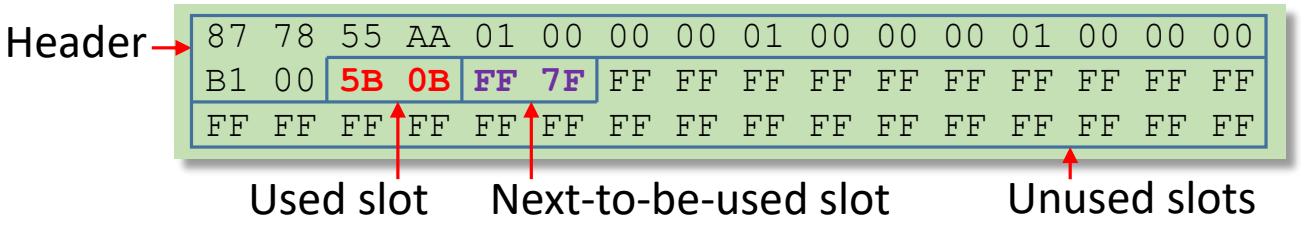
Chunk Index can be derived from
(data + crc16) by reversing CRC-16

```
#define MFS_CHUNK_SIZE 0x40
typedef struct {
  unsigned __int8 data[MFS_CHUNK_SIZE]; // Payload
  unsigned __int16 crc16; // Checksum
} T_MFS_Chunk; // 66 bytes
```

System Pages

Chunk indices stored in `axIdx` (in obfuscated form)

`axIdx[i+1]==0xFFFF` for unused slots
`axIdx[i+1]==0x7FFF` for last used slot



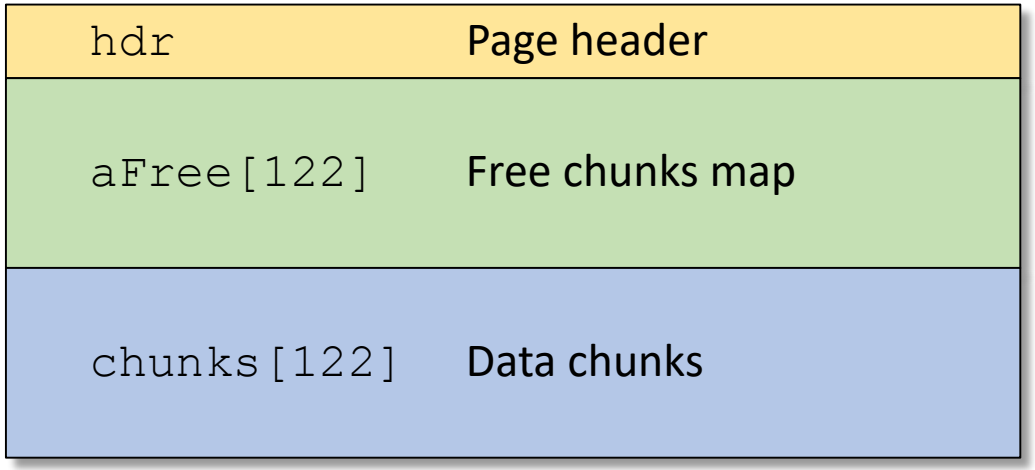
hdr	Page header
axIdx[121]	Obfuscated chunk indices
chunks[120]	System chunks

```
#define SYS_PAGE_CHUNKS 120
typedef struct {
  T_MFS_Page_Hdr hdr; // Page header
  unsigned __int16 axIdx[SYS_PAGE_CHUNKS+1]; // Obfuscated indices
  T_MFS_Chunk chunks[SYS_PAGE_CHUNKS]; // System chunks
} T_MFS_System_Page;
```

Data Pages

Stores chunks with sequential indices starting at `hdr.firstChunk`

`aFree[i]==0xFF` for unused chunks




```
#define DATA_PAGE_CHUNKS 122
typedef struct {
  T_MFS_Page_Hdr hdr; // Page header
  unsigned __int8 aFree[DATA_PAGE_CHUNKS]; // Free chunks map
  T_MFS_Chunk chunks[DATA_PAGE_CHUNKS]; // Data chunks
} T_MFS_Data_Page;
```

Data Area Reconstruction

Each Data chunk is stored exactly once

```
nDataChunks = nDataPages * 122
```

Enumerate Data pages

```
nSysChunks = min(nSysPages, pg.hdr.firstChunk)
```

Enumerate used chunks within current page

```
dataChunks[pg.hdr.firstChunk + i] = pg.chunks[i].data
```

System Area Reconstruction

Enumerate System pages in USN order

Enumerate all chunks used in the current page

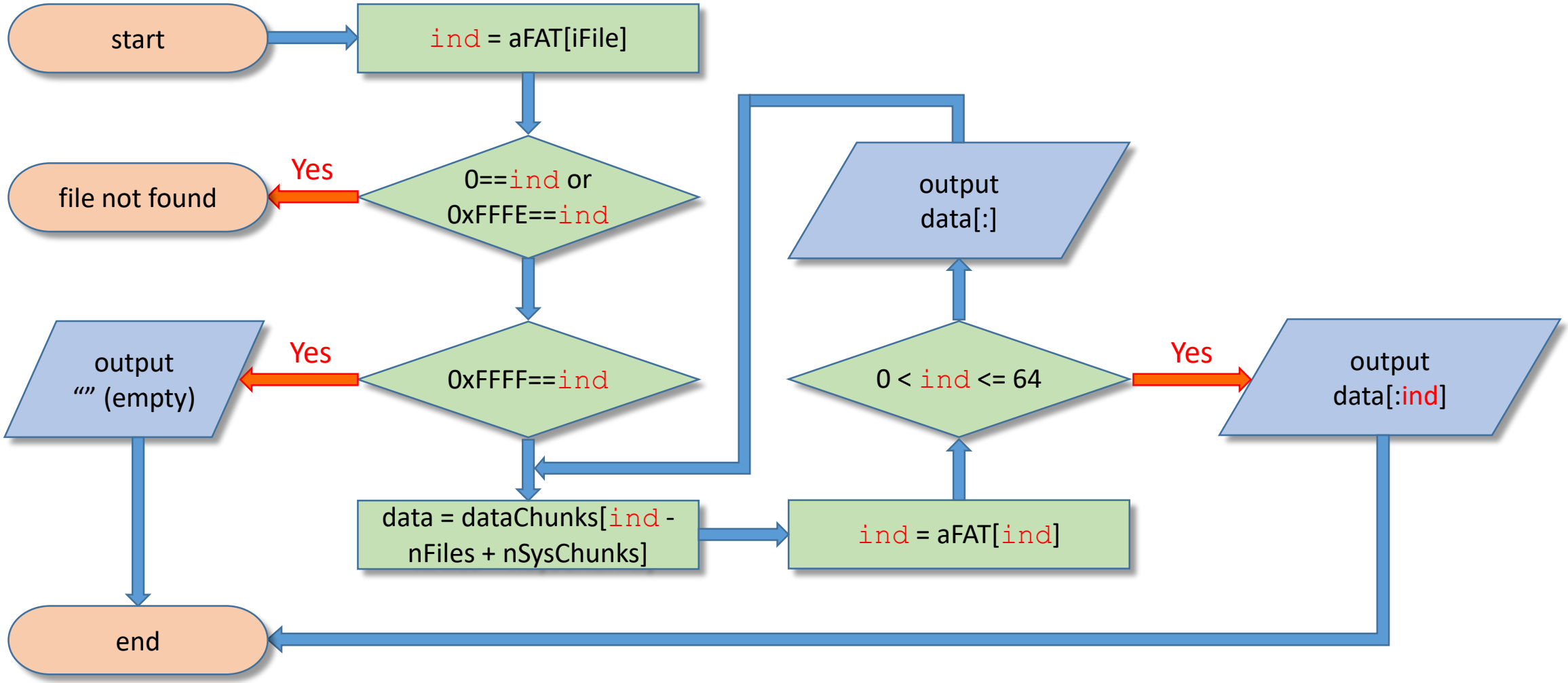
Calculate chunk Index (`iChunk`) from `pg.axIdx[i]`

`sysArea[iChunk*64 : (iChunk+1)*64] = pg.chunks[i].data`

```
typedef struct {
  unsigned __int32 sign; // Volume signature == 0x724F6201
  unsigned __int32 ver; // Volume version? == 1
  unsigned __int32 cbTotal; // Total volume capacity (System area + Data area)
  unsigned __int16 nFiles; // Number of file records
} T_MFS_Volume_Hdr; // 14 bytes

typedef struct {
  T_MFS_Volume_Hdr vol; // Volume header
  unsigned __int16 aFAT[vol.nFiles + nDataChunks]; // File Allocation Table
} T_MFS_System_Area;
```

Data Extraction from Files



MFS Templates from fit.exe

	AFS_region_256K.bin	AFS_region_400K.bin	AFS_region_1272K.bin
Total pages in MFS	32	50	159
Number of System pages	2	4	13
Number of Data pages	29	45	145
Number of System chunks	119	188	586
Number of Data chunks	3538	5490	17690
Number of file slots	256	512	1024
System area capacity (bytes)	7616	12032	37504
Data area capacity (bytes)	226432	351360	1132160

MFS Usage

Special Files

File #	Description
2, 3	AR (Anti-Replay) table
4	Used for migration after SVN (Secure Version Number) upgrade
5	File System Quota storage (related to User Info metadata extension for `vfs` module)
6	`/intel.cfg` file (default state of FS configured by Intel). SHA256 of `intel.cfg` is stored in System Info manifest extension.
7	`/fitc.cfg` file (vendor-specific FS configuration). Can be created by platform vendor using Intel's Flash Image Tool (fit.exe).
8	`/home/` directory (starting directory for ME files stored in MFS)

intel.cfg (fitc.cfg) Structure

```
typedef struct {
  char name[12]; // File name
  unsigned __int16 unused; // Always 0
  unsigned __int16 mode; // Access mode
  unsigned __int16 opt; // Deploy options
  unsigned __int16 cb; // File data length
  unsigned __int16 uid; // Owner User ID
  unsigned __int16 gid; // Owner Group ID
  unsigned __int32 offs; // File data offset
} T_CFG_Record; // 28 bytes

typedef struct {
  unsigned __int32 nRec; // Number of records
  T_CFG_Record rec[nRec]; // Records
  unsigned __int8 data[]; // File data
} T_CFG;
```

Bits	Description of `mode` fields
8..0	**rwxrwxrwx** Unix-like rights
9	**I** Enable integrity protection
10	**E** Enable encryption
11	**A** Enable anti-replay protection
13..12	**d** Record type (0: file, 1: directory)

Bits	Description of `opt` fields
0	**F** Use data from `fitc.cfg`
1	**M** Updatable by `mca` process
2..3	**?!** Unknown [for now]

*Red letters are used on the next slide

intel.cfg Partial Dump

name	mode	opt	cb	uid	gid	offset	mode	opt	path
home	11FF	0000	0000	0000	0000	00003388	d---rwxrwxrwx	----	/home/
RTFD	13C0	0009	0000	0046	0000	00003388	d--Irwx------	?--F	/home/RTFD/
..	13C0	0000	0000	0046	0000	00003388			/home/
alert_imm	136D	0001	0000	01F9	01FA	00003388	d--Ir-xr-xr-x	---F	/home/alert_imm/
AlertImm	03F8	0001	0003	01F9	01FA	00003388	--Irwxrwx---	---F	/home/alert_imm/AlertImm
..	136D	0000	0000	01F9	01FA	00003388			/home/
bup	13F9	0009	0000	0003	0115	00003388	d--Irwxrwx--x	?--F	/home/bup/
bup_sku	13C0	0009	0000	0003	0000	00003388	d--Irwx------	?--F	/home/bup/bup_sku/
emu_fuse_map	01A0	0009	0000	0003	00EE	0000338B	---rw-r-----	?--F	/home/bup/bup_sku/emu_fuse_map
fuse_ip_base	01A0	0009	0000	0003	00EE	0000338B	---rw-r-----	?--F	/home/bup/bup_sku/fuse_ip_base
plat_n_sku	01A0	0009	0000	0003	00EE	0000338B	---rw-r-----	?--F	/home/bup/bup_sku/plat_n_sku
..	13C0	0000	0000	0003	0000	00003388			/home/
ct	01E0	0009	0000	0003	015F	0000338B	---rwxr-----	?--F	/home/bup/ct
df_cpu_info	01FF	0009	0004	0003	00CE	0000338B	---rwxrwxrwx	?--F	/home/bup/df_cpu_info
invokemebx	01B0	0009	0004	0003	0115	0000338F	---rw-rw----	?--F	/home/bup/invokemebx
mbp	01A0	0009	0004	0003	00CE	00003393	---rw-r-----	?--F	/home/bup/mbp
si_features	01A0	0009	0014	0003	015F	00003397	---rw-r-----	?--F	/home/bup/si_features
..	13F9	0000	0000	0003	0115	00003388			/home/
gpio	13F8	0009	0000	0003	0190	00003388	d--Irwxrwx---	?--F	/home/gpio/
csme_pins	01B0	0009	0028	0003	0190	000033AB	---rw-rw----	?--F	/home/gpio/csme_pins
..	13F8	0000	0000	0003	0190	00003388			/home/
h_res_w	13FF	0001	0000	01FF	01FF	00003388	d--Irwxrwxrwx	---F	/home/h_res_w/
hrw_conf	03FF	0001	0000	01F8	01F8	000033D3	--Irwxrwxrwx	---F	/home/h_res_w/hrw_conf
..	13FF	0000	0000	01FF	01FF	00003388			/home/
hm	136D	0001	0000	0205	0208	00003388	d--Ir-xr-xr-x	---F	/home/hm/
exceptions	13ED	0001	0000	0205	0208	00003388	d--Irwxr-xr-x	---F	/home/hm/exceptions/

MFS Directory

```
typedef struct {
  unsigned __int32 fileno; // iFS,salt,iFile
  unsigned __int16 mode; // Access mode
  unsigned __int16 uid; // Owner User ID
  unsigned __int16 gid; // Owner Group ID
  unsigned __int16 salt; // Another salt
  char name[12]; // File name
} T_MFS_Folder_Record; // 24 bytes
```

Bits	Description of `fileno` fields
11..0	iFile (0..4095)
27..12	16 bits of salt
31..28	FileSystem ID (always 1)

Bits	Description of `mode` fields
8..0	**rwxrwxrwx** Unix-like rights
9	**I** Enable integrity protection
10	**E** Enable encryption
11	**A** Enable anti-replay protection
13	**N** Use non-Intel keys
15..14	**d** Record type (0: file, 1: directory)

Dump of `home/policy/pwdmgr/` directory

iFile	fileno	mode	uid	gid	salt	name	size
105:	1F5BC105	dN---Irwxrwx---	0055	00EE	A84D	.	<dir>
0F6:	14EBD0F6	dN---Irwxrwx--x	0055	0115	410C	..	<dir>
107:	10000107	-----rw-------	0055	0000	0000	**maxattempts**	0
108:	10000108	-----rw-r-----	0055	00EE	0000	**pwdpolicy**	0
109:	1DE0C109	N--EIrw-rw----	0055	00EE	C098	**segreto**	11
10A:	1000010A	-----rw-------	0055	0000	0000	**sendpwd**	0

Integrity, Encryption, Anti-Replay

If **I** bit is set, raw file contains additional security blob at the end (52 bytes in length)

Integrity protection also enabled and mandatory for:

- AR tables (iFile == 2, 3)
- `/home/` directory (iFile == 8)

```
typedef struct {
  unsigned __int8 hmac[32]; // HMAC value
  unsigned __int32 antiReplay:2; // Anti-Replay
  unsigned __int32 encryption:1; // Encryption
  unsigned __int32 unk7:7;
  unsigned __int32 iAR:10; // Index in AR table
  unsigned __int32 unk12:12;
  union {
    struct ar { // Anti-Replay data
      unsigned __int32 rnd; // AR Random value
      unsigned __int32 ctr; // AR Counter value
    };
    unsigned __int8 nonce[16]; // AES-CTR nonce
  };
} T_FileSecurity; // 52 bytes
```

HMAC covers file data, security blob (with `hmac` zeroed), `fileno` and `salt` (from directory)

Additional Info

FS Security Keys

There are up to 10 keys involved in FS Security

Intel Integrity	Non-Intel Integrity
Current keys (for current SVN)	
Intel Confidentiality	Non-Intel Confidentiality

Intel Integrity	Non-Intel Integrity
Previous* keys (optional)	
Intel Confidentiality	Non-Intel Confidentiality

*Previous keys are calculated if current SVN > 1 and PSVN partition contains valid data. These keys are used for migrating files created before the SVN was updated.

RPMC HMAC #0	RPMC HMAC #1

Replay-Protected Monotonic Counter (RPMC) is optional feature of SPI Flash chip

Crypto Engine / Usage Practices

Features

- HW Engines for AES, RSA, Hash/HMAC
- Secure Key Storage (SKS)
  - Keys 1..11 are 128 bits long
  - Keys 12..21 are 256 bits long
  - Keys can be used by AES/HMAC
  - Keys cannot be extracted
- Direct access to HW Engines/SKS allowed for `ROM`, `bup`, and `crypto` only

Usage

HMAC Key and Wrapping Key are loaded into SKS

To prepare the necessary key:

- Derive it with HMAC*
- Wrap it with AES and store in mem
- Wipe plaintext key

To use wrapped key:

- Unwrap it with AES into SKS
- Use AES/HMAC with SKS linkage

* This is the only moment when the Plaintext Key is available in memory (until wiped)

Key Derivation and Usage

VFS Confidentiality/Integrity key	Intel	Non-Intel
Never stored on Flash in any form	Yes	Yes
Persists in memory in wrapped form only (SKS key #21)	Yes	Yes
Cannot be unwrapped to memory (SKS only)	Yes	Yes
Depends on SVN value (1-byte)	Yes	Yes
Depends on secret obtained from GEN device	Yes	Yes
Copy of GEN secret wiped in `ROM` (before passing control to `rbe`)	Yes	Yes
GEN device reading disabled by `ROM` (before passing control to `rbe`)	Yes	Yes
GEN secret unavailable under JTAG	Yes	**No**

Note: Rare module protects files with Intel keys:
`sigma`, `ptt`, `dal_ivm`, `mca`

File System Types in VFS

iFS	Name	Description
0	root	Defined in `vfs`. Can hold up to 1024 entries. Initially contains `/`, `/dev/`, `/etc/`, `/etc/rc`, `/temp/`
1	home	Handles files from MFS, supports security features.
2	bin	Maps modules from Code Partition Directory ($CPD).
3	susram	Defined in `bup` and `vfs`. Uses 3072 bytes of NV Suspend RAM.
4	fpf	Defined if `fpf`. Not available in Server Platform Services firmware.
5	dev	Maps devices from Special File Producer metadata extension.
6	umafs	Never seen any references to this…

Black Hat Sound Bytes

1. Physical access (to SPI chip) allows R/W access to ME Flash File System content (as raw files). `fitc.cfg` can also be modified in an arbitrary way.

2. Intel has developed a sophisticated and flexible security model to protect against various types of attacks on data-at-rest.

3. Knowing the GEN secret for non-Intel keys (just 16 bytes) permits R/W access to most data stored in MFS (for any SVN). Code execution in `bup` permits access to everything (for current SVN) by re-calculating keys.

Thanks! Questions?

Intel ME:
Flash File System Explained

POSITIVE TECHNOLOGIES