

RUCTF 2018

13-14 апреля 2018

Екатеринбург, Россия

Intel ME: Детали устройства файловой системы в Flash-памяти

POSITIVE TECHNOLOGIES

ptsecurity.com

Максим Горячий

MGoryachy@ptsecurity.com

Марк Ермолов

MErmolov@ptsecurity.com

Дмитрий Скляр

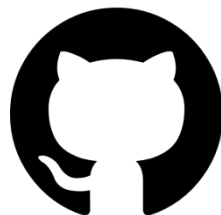
DSklyarov@ptsecurity.com

- Intel ME: Известные факты
- Предварительная работа
 - Первые шаги. Разбираем прошивку, извлекаем метаданные
 - Huffman Encoding. Восстановление таблиц
- ME и хранение данных
 - Flash-память. Физические особенности
 - Внутренности MFS. Как организована файловая система
 - Использование MFS. Как Intel ME работает с файлами
- Дополнительная информация
 - Особенности применения криптографии в ME 11

Intel ME

Известные факты

- 2004: первая версия AMT (Active Management Technology)
 - процессор встроен в сетевой контроллер Intel 82573E
- 2006: AMT v2. Появление термина ME (Management Engine)
 - процессор с архитектурой ARCompact, встроен в чипсет
- 2010: ME v6
 - процессор с архитектурой ARCompact, OS ThreadX
- 2015: CSME (Converged Security & Management Engine) v11
 - процессор с архитектурой x86



skochinsky / me-tools

Intel ME Secrets

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

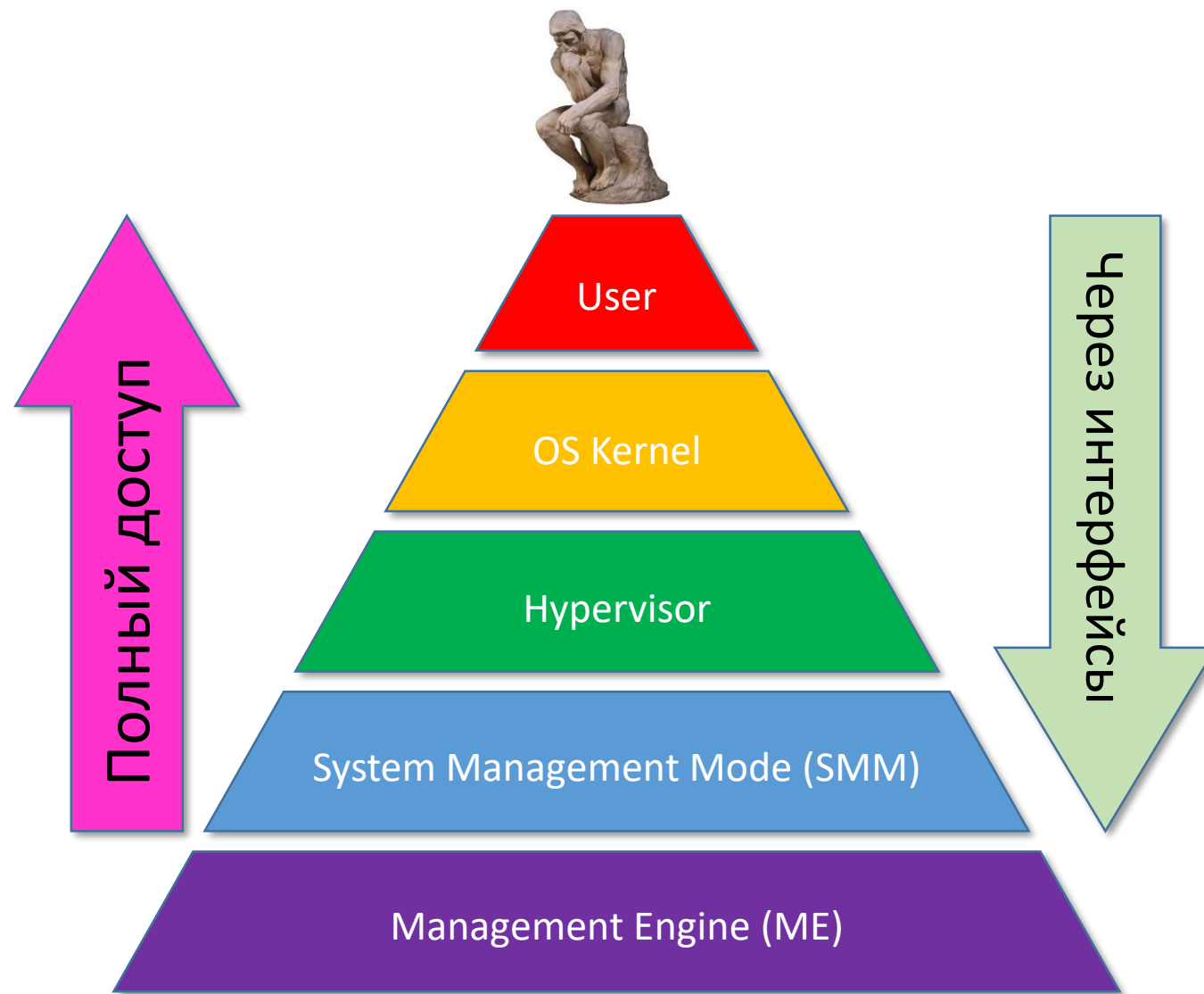
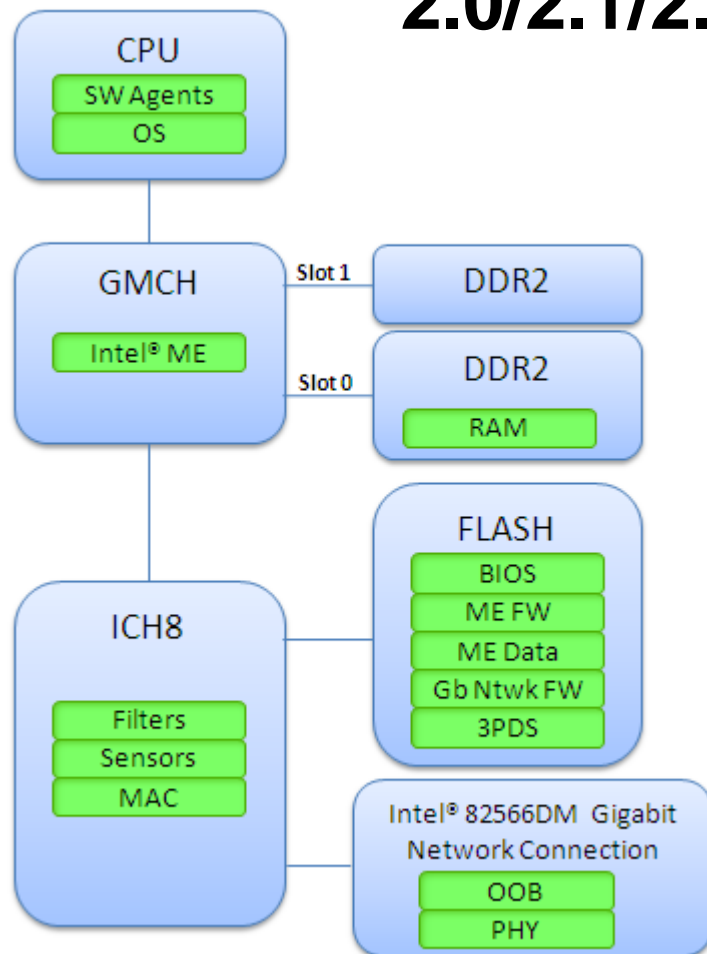
Igor Skochinsky
Hex-Rays

RECON 2014
Montreal



Patents / White Papers /
Documentation

Архитектура Intel AMT 2.0/2.1/2.2



- Внеполосный доступ к сетевому интерфейсу (LAN и WLAN)
- [KVM](#) (Keyboard Video Mouse) – удаленное управление
- [SOL](#) (Serial Over LAN) и [Storage Redirection](#) (USB-R и IDE-R)
- Другие [ВОЗМОЖНОСТИ](#):
[Access Monitor](#), [Agent Presence](#), [Alarm Clock](#), [Discovery](#), [Event Manager](#),
[Fast Call for Help](#), [Firmware Update](#), [General Info](#), [Hardware Asset](#),
[Remote Access](#), [Remote Control](#), [Storage Administration and Operations](#),
[System Defense](#), [User Consent](#), [WS-Eventing](#), ...

Первые шаги

Разбираем прошивку, извлекаем метаданные

```
C:\MEU>meu.exe -gen CodePartition  
  
Saving XML ...  
XML file written to CodePartition.xml
```

C:\MEU\CodePartition.xml

```
<CPModules>  
  <CPDataModule name="ish_main" enabled="true">  
    <InputFile value="ish_main.bin" />  
    <CompressionType value="LZMA" value_list="NOT_COMPRESSED,,LZMA" />  
    <ProcessId value="0xf6" />  
  </CPDataModule>  
</CPModules>
```

Строка не используется!

```
dd offset aNot_compressed ; "NOT_COMPRESSED"  
dd offset aHuffman       ; "HUFFMAN"  
dd offset aLzma           ; DATA XREF: sub_246AF0+506↑r  
                          ; "LZMA"  
dd offset aInvalid_compre ; "INVALID_COMPRESSION_SETTING"
```

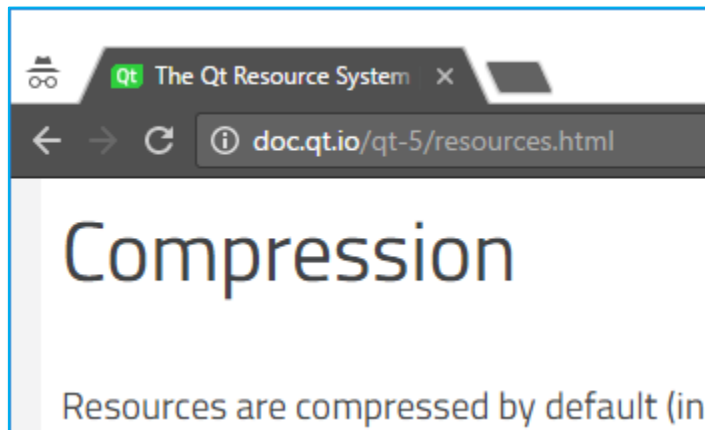
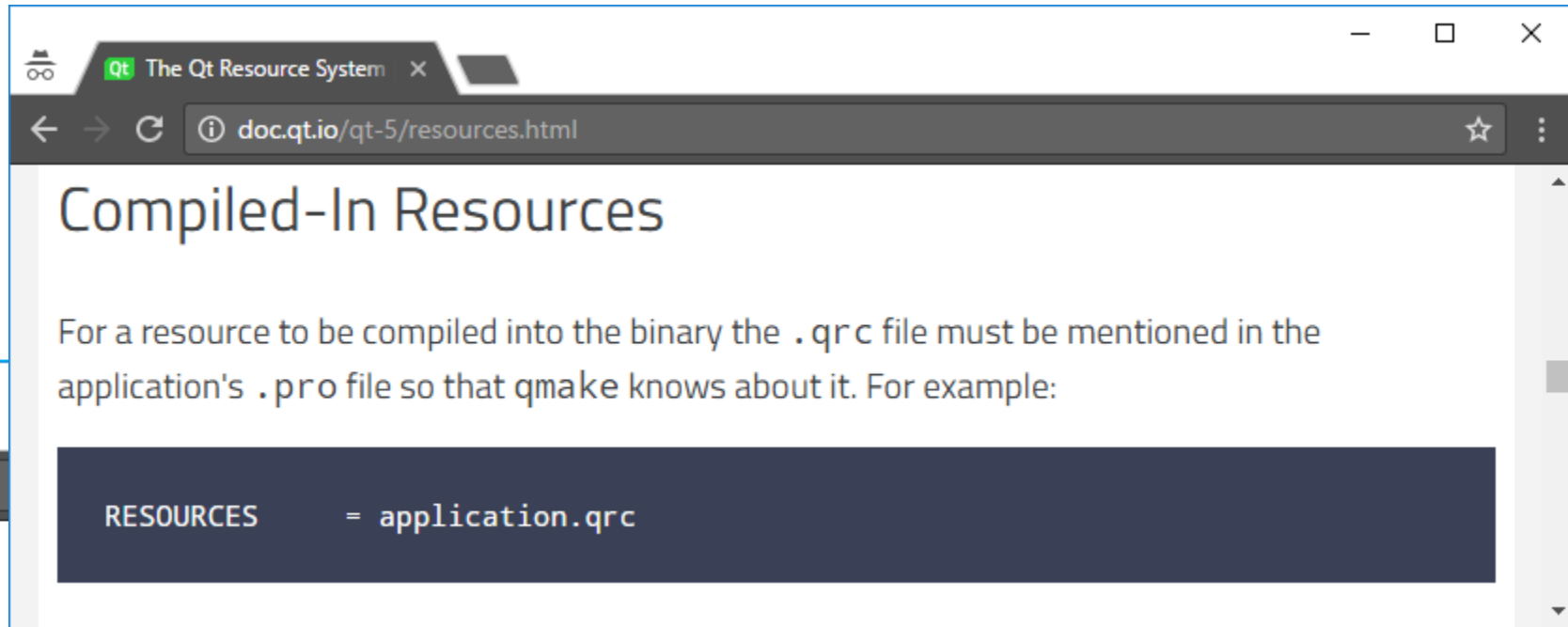
```
C:\MEU>python C:\Python27\Scripts\binwalk meu.exe
```

DECIMAL	HEXADECIMAL	DESCRIPTION

0	0x0	Microsoft executable, portable (PE)
2810520	0x2AE298	XML document, version: "1.0"
2842816	0x2B60C0	Copyright string: "Copyright (c) "
2851456	0x2B8280	Zlib compressed data, default compression
2858473	0x2B9DE9	XML document, version: "1.0"
2860580	0x2BA624	Zlib compressed data, default compression
2867878	0x2BC2A6	Zlib compressed data, default compression
		...



```
db 'QResourceInfo: Resource [%s] has both data and children!','0
    ; DATA XREF: sub_3DB140+1FE↑o
align 10h
dd offset unk_6283FC
dd offset sub_3D93E0 ; DATA XREF: sub_3D8FD0+44↑o
db 'QResourceFileEngine::open: Missing file name',0
```



Flash Image Tool (`fit.exe`):

- `:/res/bin/AFS_region_1272K.bin`
- `:/res/bin/AFS_region_256K.bin`
- `:/res/bin/AFS_region_400K.bin`
- `:/res/bin/descriptor_template.bin`
- `:/res/ftool/spt_layout.xsd`
- `:/res/xml/bxt_fit_cfg_dflt.xml`
- `:/res/xml/bxt_fit_layout.xml`
- `:/res/xml/spt_fit_cfg_dflt.xml`
- `:/res/xml/spt_fit_cfg_dflt_H.xml`
- `:/res/xml/spt_fit_layout.xml`

`:/res/xml/spt_fit_layout_common.xml`

`:/res/xml/spt_fit_layout_H.xml`

`:/res/xml/spt_ftool_cfg_dflt.xml`

`:/res/xml/spt_ftool_layout.xml`

`:/res/xml/spt_layout.xsd`

Manifest Extension Utility (`meu.exe`):

`:/res/xml/bxt_meu_cfg_dflt.xml`

`:/res/xml/bxt_meu_layout.xml`

`:/res/xml/spt_meu_cfg_dflt.xml`

`:/res/xml/spt_meu_layout.xml`

А вы читали “Modern Operating Systems”?

POSITIVE TECHNOLOGIES

```
> strings vfs
...
..\..\src\os\servers\vfs\misc.c
FS: bogus child for forking
FS: forking on top of in-use child
...
```

MINIX3 by Andrew Tanenbaum

```
Directory of minix3-master\servers\vfs

14.03.2010  23:52    14'978  main.c
14.03.2010  23:52         741  Makefile
14.03.2010  23:52    17'653  misc.c
14.03.2010  23:52         677  mmap.c
14.03.2010  23:52    15'650  mount.c
...
```

"FS: bogus child for forking"

All

Images

Videos

News

Shopping

More

Settings

Tools

6 results (0.34 seconds)

[misc.c in minix-filesystem | source code search engine - Searchcode](https://searchcode.com/codesearch/view/55926734/)

<https://searchcode.com/codesearch/view/55926734/>

childno = _ENDPOINT_P(m_in.child_endpt); if(childno < 0 || childno >= NR_PROCS) panic(__FILE__, "FS: bogus child for forking", m_in.child_endpt); ...

```
/* PM gives child endpoint, which implies process slot information.
 * Don't call isokendpt, because that will verify if the endpoint
 * number is correct in fproc, which it won't be.
 */
childno = _ENDPOINT_P(m_in.child_endpt);
if(childno < 0 || childno >= NR_PROCS)
    panic(__FILE__, "FS: bogus child for forking", m_in.child_endpt);
if(fproc[childno].fp_pid != PID_FREE)
    panic(__FILE__, "FS: forking on top of in-use child", childno);
```

Huffman Encoding

Восстановление таблиц

- Два набора таблиц (для разных версий ME)
 - Длины кодовых слов [7..19] и [7..15] бит
- Используется Canonical Huffman Coding
- Каждое кодовое слово кодирует [1..15] байт
- Для одной версии ME – две таблицы (Code и Data)
 - Длины закодированных значений для любого кодового слова совпадают в обеих таблицах

Codeword Len	Highest Codeword	Lowest Codeword	Count
7	1111111	1110111	9
8	11101101	10100011	75
9	101000101	010111101	137
10	0101111001	0011001011	175
11	00110010101	00011010111	191
12	000110101101	000011011101	209
13	0000110111001	0000011001001	241
14	00000110010001	00000001010000	322
15	000000010011111	0000000000000000	160

- Диапазоны длин кодовых слов
- Границы значений кодовых слов одной длины (Shape)
- Длины закодированных последовательностей для каждого кодового слова (Length)
- Закодированные значения для каждого кодового слова в обеих таблицах (Value)

Делим упакованные данные на страницы

kernel.huff	↓FRO -----	+00000050
00000000:	00 00 00 40-80 0D 00 40-80 1E 00 40-80 2C 00 40	
00000010:	80 3A 00 40-00 48 00 40-00 55 00 40-C0 62 00 40	
00000020:	C0 70 00 40-00 7F 00 40-00 8D 00 40-C0 9A 00 40	
00000030:	00 A9 00 40-40 B7 00 40-00 C5 00 40-00 D3 00 40	
00000040:	40 E1 00 40-40 EF 00 40-80 F7 00 40-40 FA 00 C0	
00000050:	48 71 26 D6-C2 73 AB 9C-7D CF 71 39-F8 F9 E8 57	
00000060:	DA BC 67 DC-F5 8F AB C6-7D CF 70 F8-97 3D 4C 64	

Code @0000	Code @8D00
Code @0D80	Code @9AC0
Code @1E80	Code @A900
Code @2C80	Code @B740
Code @3A80	Code @C500
Code @4800	Code @D300
Code @5500	Code @E140
Code @62C0	Code @EF40
Code @70C0	Code @F780
Code @7F00	Data @FA40

- Номер таблицы в двух старших битах: 0b01==Code, 0b11==Data
- Младшие 30 бит - смещение сжатых данных (от конца таблицы)
- Конец сжатых данных явно не задан
- Неиспользуемые области заполнены нулевыми битами
- Распакованная страница занимает 4096 байт

$[0xB2] + [0xEC]*273$

Длина последовательности в хвосте: $4096/273 == 15$

Длина последовательности в начале: $4096-273*15 == 1$

111011001110110011101100
011101100111011001110110
001110110011101100111011

kernel.huff	↓FRO	-----	+0000FA40
0000FA00:	9E	9E 9E 9E 9E-9E 9E 9E 9E-9E 9E 9E 9E-9E 9E 9E 9E	
0000FA10:	9E	98 00 00-00 00 00 00-00 00 00 00-00 00 00 00	
0000FA20:	00	00 00 00-00 00 00 00-00 00 00 00-00 00 00 00	
0000FA30:	00	00 00 00-00 00 00 00-00 00 00 00-00 00 00 00	
0000FA40:	B2	EC EC EC-EC EC EC EC-EC EC EC EC-EC EC EC EC	
0000FA50:	EC	EC EC EC-EC EC EC EC-EC EC EC EC-EC EC EC EC	
0000FA60:	EC	EC EC EC EC-EC EC EC EC-EC EC EC EC-EC EC EC EC	
0000FA70:	EC	EC EC EC EC-EC EC EC EC-EC EC EC EC-EC EC EC EC	
0000FA80:	EC	EC EC EC EC-EC EC EC EC-EC EC EC EC-EC EC EC EC	
0000FA90:	EC	EC EC EC EC-EC EC EC EC-EC EC EC EC-EC EC EC EC	
0000FAA0:	EC	EC EC EC EC-EC EC EC EC-EC EC EC EC-EC EC EC EC	
0000FAB0:	EC	EC EC EC EC-EC EC EC EC-EC EC EC EC-EC EC EC EC	
0000FAC0:	EC	EC EC EC EC-EC EC EC EC-EC EC EC EC-EC EC EC EC	
0000FAD0:	EC	EC EC EC EC-EC EC EC EC-EC EC EC EC-EC EC EC EC	
0000FAE0:	EC	EC EC EC EC-EC EC EC EC-EC EC EC EC-EC EC EC EC	
0000FAF0:	EC	EC EC EC EC-EC EC EC EC-EC EC EC EC-EC EC EC EC	
0000FB00:	EC	EC EC EC EC-EC EC EC EC-EC EC EC EC-EC EC EC EC	
0000FB10:	EC	EC EC EC EC-EC EC EC EC-EC EC EC EC-EC EC EC EC	
0000FB20:	EC	EC EC EC EC-EC EC EC EC-EC EC EC EC-EC EC EC EC	
0000FB30:	EC	EC EC EC EC-EC EC EC EC-EC EC EC EC-EC EC EC EC	
0000FB40:	EC	EC EC EC EC-EC EC EC EC-EC EC EC EC-EC EC EC EC	
0000FB50:	EC EC	00 00-00 00 00 00-00 00 00 00-00 00 00 00	
0000FB60:	00	00 00 00-00 00 00 00-00 00 00 00-00 00 00 00	

- В разных прошивках модули с одинаковыми именами могут быть упакованы как LZMA так и Huffman Encoding
- В *Module Attributes Extension* есть SHA256 от модуля
 - Но нет ни одного совпадения для разных алгоритмов сжатия
- Хеш модулей, сжатых LZMA, считается от упакованных данных!
- Распаковываем LZMA, считаем SHA256, сравниваем...
- Получаем множество пар для анализа!

- Определение Shape (границ длин кодовых слов) – плохо формализуемый процесс, ручной метод проб и ошибок
- Восстановление Length (длины для каждой закодированной последовательности) – решение СЛАУ методом Гаусса
 - Одно уравнение для одной страницы
 - Можно мешать страницы кода и данных, открытые тексты не нужны
 - Неизвестные – длины закодированных последовательностей
 - Коэффициенты – сколько раз встретилось кодовое слово
 - Свободный член – 4096
- Значения закодированных последовательностей - тривиально

- Известны длины для 92% последовательностей
- Восстановлено 70% последовательностей для кодовой таблицы
- Восстановлено 68% последовательностей для таблицы данных
- Остается некоторая неопределенность в Shape

- *Module Attributes Extension* содержит SHA256 от распакованных данных
- Можно автоматически подбирать неизвестные байты
 - Чем ближе к концу файлу тем быстрее
 - Почти моментально для 1 и 2 байт
 - Порядка часа для 3 байт, несколько дней для 4 байт
 - Легко распараллелить на несколько ядер/машин
- Подбирать 5 и более байт почти бессмысленно :(

- Найти такое же место, сжатое другой таблицей
- Проанализировать все вхождения последовательности, построить систему ограничений и найти решение
- Найти повторяющиеся (в разных модулях) константы
- Вычислить смещения до функций или констант
- Угадать строковые константы (или подглядеть в OpenSource)
- Воссоздать куски кода из открытых библиотек
- Найти похожую функцию в других версиях модуля
- Найти похожую функцию в предыдущих версиях ME

- Восстановлено 89.4% последовательностей кодовой таблицы
- Восстановлено 86.4% последовательностей таблицы данных
- Можем распаковать любой модуль ME 11 из тех, что доступны
- 19 июня 2017 года IllegalArgument публикует на GitHub фрагменты таблиц (с покрытием 80.8% и 78.1% соответственно)

- 17 июля 2017 года Марк Ермолов и Максим Горячий получили возможность напрямую обращаться к Huffman Decoder
- Подготовили 4 сжатых страницы (по две для кода и для данных)
- Восстановили все 1519 последовательностей для обеих таблиц
- В таблице для данных значение 00410E088502420D05 соответствует как последовательности 10100111 (длиной 8 бит), так и последовательности 000100101001 (длиной 12 бит)

Flash-память

Физические особенности

- Изобретена в 1984
- Две разновидности:
 - NOR (1988, Intel)
 - NAND (1989, Toshiba)
- Сохраняет электрический заряд в транзисторе с плавающим затвором
- Обеспечивает доступность данных в течение 10-100 лет



Intel's m-SATA 80G SSD (2010)

- Любой байт можно записывать независимо
- Необходимо стирание (все биты в 1) перед перезаписью
- Стирается только блок целиком (например, 8K)
- Гарантированное число циклов стирания невелико
 - Обычно между 10'000 и 1'000'000
 - Нестираемые блоки помечаются как “плохие” и не используются

- Минимизация числа стираний

Инкрементальное изменение данных во избежание лишних стираний

- Выравнивание износа

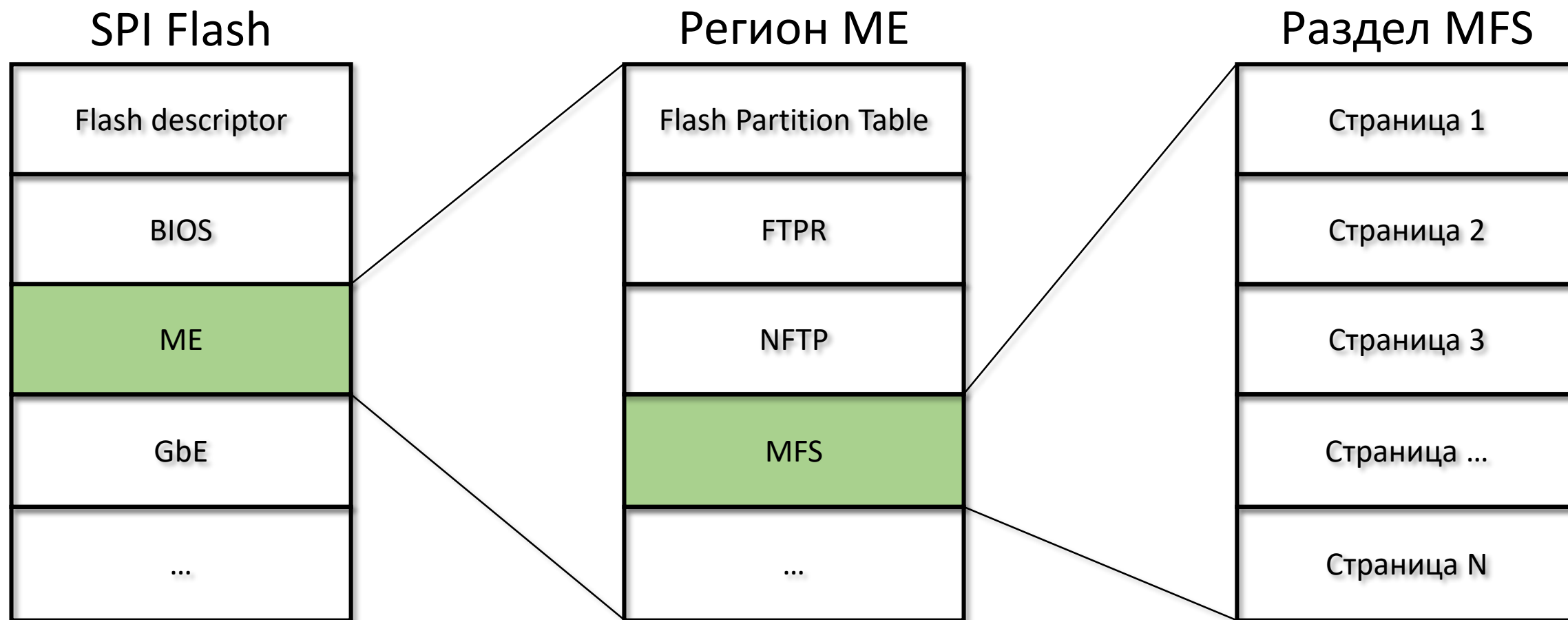
Максимально равномерное распределение стираний между блоками

Популярные файловые системы на Flash, используемые в Linux:

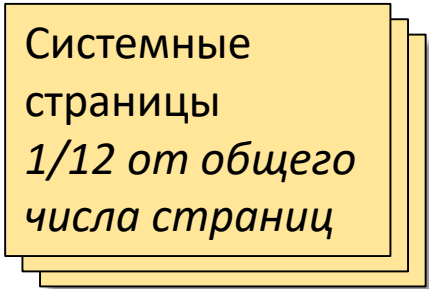
- JFFS, JFFS2 и YAFFS
- UBIFS
- LogFS
- F2FS

Внутренности MFS

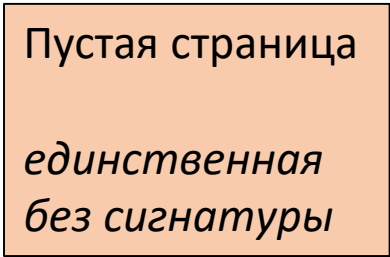
Как организована файловая система



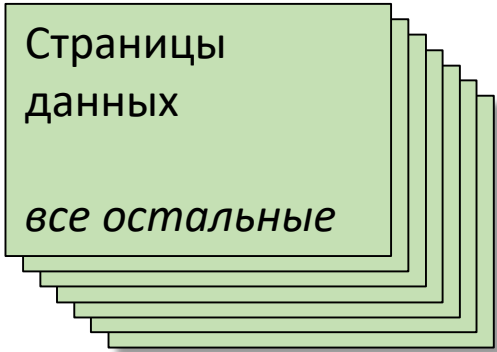
MFS представляет собой набор страниц фиксированного размера
(8192 == 0x2000 байт каждая)



Системные
страницы
*1/12 от общего
числа страниц*



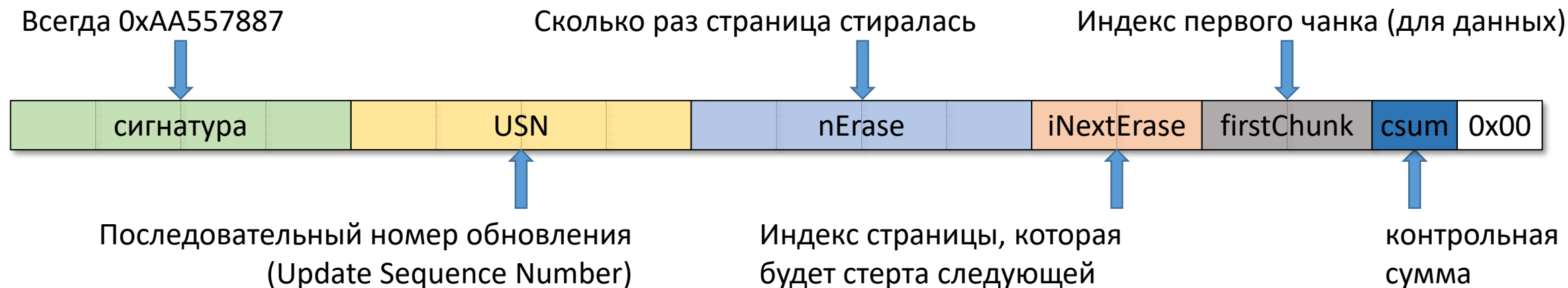
Пустая страница
*единственная
без сигнатуры*



Страницы
данных
все остальные

```
#define MFS_PAGE_SIZE 0x2000
cbMFS = sizeof(MFS); // Размер раздела MFS
nPages = cbMFS / MFS_PAGE_SIZE; // Всего страниц в MFS

nSysPages = nPages / 12; // Число системных страниц
nDataPages = nPages - nSysPages - 1; // Страниц данных
```



```
typedef struct {
    unsigned __int32 signature; // Сигнатура страницы == 0xAA557887
    unsigned __int32 USN; // Последовательный номер обновления
    unsigned __int32 nErase; // Сколько раз страница стиралась
    unsigned __int16 iNextErase; // Индекс следующей стираемой станицы
    unsigned __int16 firstChunk; // Индекс первого чанка (для данных)
    unsigned __int8 csum; // Контрольная сумма первых 16 байт заголовка
    unsigned __int8 b0; // Всегда 0
} T_MFS_Page_Hdr; // 18 байт
```

Отдельный чанк (66 байт)



*СCITT CRC-16 вычисляется от данных чанка и 16-битового (2-байтового) индекса чанка

Индекс чанка можно получить из
(data + crc16) путем обращения CRC-16

Чанк #
0x1201

Чанк #
0x1202

Чанк #
0x1203

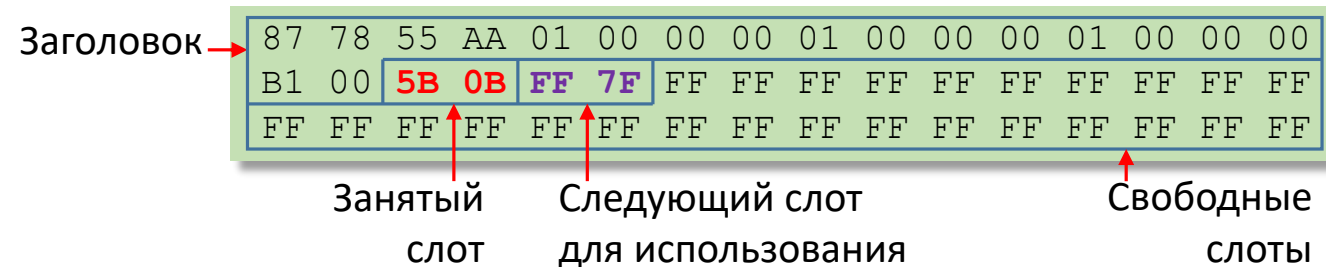
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
F4	D4	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	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	A7	81	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	96	B2	00	00	00	00	00	00	00	00	00	00	00

```
#define MFS_CHUNK_SIZE 0x40
typedef struct {
    unsigned __int8 data[MFS_CHUNK_SIZE]; // Данные
    unsigned __int16 crc16; // Контрольная сумма
} T_MFS_Chunk; // 66 байт
```

Индексы чанков хранятся в `axIdx` (в обфусцированном виде)

`axIdx[i+1]==0xFFFF` для неиспользуемых слотов

`axIdx[i+1]==0x7FFF` для последнего занятого слота



hdr	Заголовок страницы
axIdx[121]	Индексы чанков
chunks[120]	Системные чанки

```
#define SYS_PAGE_CHUNKS 120
typedef struct {
    T_MFS_Page_Hdr hdr; // Заголовок страницы
    unsigned __int16 axIdx[SYS_PAGE_CHUNKS+1]; // Индексы чанков
    T_MFS_Chunk chunks[SYS_PAGE_CHUNKS]; // Системные чанки
} T_MFS_System_Page;
```

Хранят чанки с последовательными индексами начиная с `hdr.firstChunk`

`aFree[i] == 0xFF` для неиспользуемых чанков

<code>hdr</code>	Заголовок страницы
<code>aFree[122]</code>	Карта свободных чанков
<code>chunks[122]</code>	Чанки данных

```
#define DATA_PAGE_CHUNKS 122
typedef struct {
    T_MFS_Page_Hdr hdr; // Заголовок страницы
    unsigned __int8 aFree[DATA_PAGE_CHUNKS]; // Карта свободных чанков
    T_MFS_Chunk chunks[DATA_PAGE_CHUNKS]; // Чанки данных
} T_MFS_Data_Page;
```

Каждый чанк данных сохранен ровно 1 раз

```
nDataChunks = nDataPages * 122
```

Перебираем все страницы данных

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

Перебираем все использованные чанки на текущей странице

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

Перебираем системные страницы в порядке возрастания USN

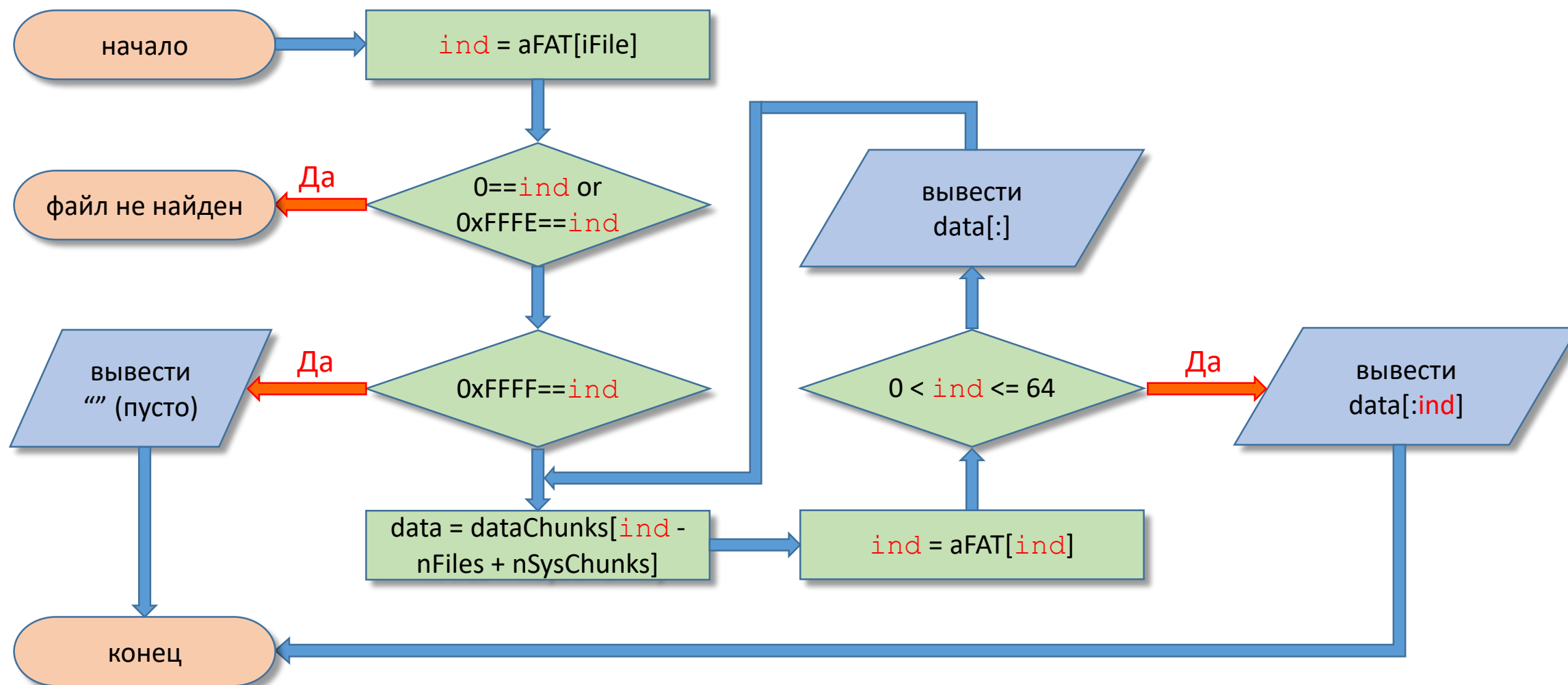
Перебираем все использованные чанки на текущей странице

Вычисляем индекс чанка (iChunk) на основе pg.axIdx[i]

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

```
typedef struct {
    unsigned __int32 sign; // Сигнатура тома == 0x724F6201
    unsigned __int32 ver; // Версия тома? == 1
    unsigned __int32 cbTotal; // Общая емкость тома (системная область + данные)
    unsigned __int16 nFiles; // Число файловых записей
} T_MFS_Volume_Hdr; // 14 байт

typedef struct {
    T_MFS_Volume_Hdr vol; // Заголовок тома
    unsigned __int16 aFAT[vol.nFiles + nDataChunks]; // Таблица размещения файлов
} T_MFS_System_Area;
```

	AFS_region_256K.bin	AFS_region_400K.bin	AFS_region_1272K.bin
Всего страниц в MFS	32	50	159
Число системных страниц	2	4	13
Число страниц данных	29	45	145
Число системных чанков	119	188	586
Число чанков данных	3538	5490	17690
Число файловых записей	256	512	1024
Емкость системной области (байт)	7'616	12'032	37'504
Емкость области данных (байт)	226'432	351'360	1'132'160

Использование MFS

Как Intel ME работает с файлами

#файла	Описание
2,3	Таблицы для защиты от атак повторения (Anti-Replay tables)
4	Используется для миграции после обновления SVN*
5	Информация об использовании квот на файловую систему (связано с User Info metadata extension для модуля vfs)
6	Файл /intel.cfg (начальное состояние ФС заданное Intel) SHA256 от intel.cfg хранится в System Info manifest extension
7	Файл/fitc.cfg (состояние ФС заданное вендором) Создается при помощи Intel's Flash Image Tool (fit.exe)
8	Директория/home/ (стартовая для файлов, хранимых в MFS)

*Secure Version Number (SVN) увеличивается в случае исправления серьезных уязвимостей для препятствования откату к уязвимой версии

Структура файла intel.cfg (fitc.cfg)

```
typedef struct {
    char name[12]; // Имя файла
    unsigned __int16 unused; // Всегда 0
    unsigned __int16 mode; // Режим доступа
    unsigned __int16 opt; // Опции развертывания
    unsigned __int16 cb; // Размер данных файла
    unsigned __int16 uid; // User ID владельца
    unsigned __int16 gid; // Group ID владельца
    unsigned __int32 offs; // Смещение данных
} T_CFG_Record; // 28 байт

typedef struct {
    unsigned __int32 nRec; // Число записей
    T_CFG_Record aRec[nRec]; // Сами записи
    unsigned __int8 data[]; // Данные файлов
} T_CFG;
```

Биты	Состав поля <i>mode</i>
8..0	rwxrwxrwx Unix-like права
9	I Integrity (целостность)
10	E Encryption (шифрование)
11	A Anti-Replay
13..12	d Тип (0:файл,1:директория)

Биты	Состав поля <i>opt</i>
0	F можно задать в fitc.cfg
1	M Можно обновлять через mca
2..3	?! Неизвестно [пока]

***Красные** буквы встретятся
на следующем слайде

Фрагмент дампа intel.cfg

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--lrwx-----	?--F	/home/RTFD/
..	13C0	0000	0000	0046	0000	00003388			/home/
alert_imm	136D	0001	0000	01F9	01FA	00003388	d--lr-xr-xr-x	---F	/home/alert_imm/
AlertImm	03F8	0001	0003	01F9	01FA	00003388	--lrwxrwx---	---F	/home/alert_imm/AlertImm
..	136D	0000	0000	01F9	01FA	00003388			/home/
bup	13F9	0009	0000	0003	0115	00003388	d--lrwxrwx--x	?--F	/home/bup/
bup_sku	13C0	0009	0000	0003	0000	00003388	d--lrwx-----	?--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--lrwxrwx---	?--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--lrwxrwxrwx	---F	/home/h_res_w/
hrw_conf	03FF	0001	0000	01F8	01F8	000033D3	--lrwxrwxrwx	---F	/home/h_res_w/hrw_conf
..	13FF	0000	0000	01FF	01FF	00003388			/home/
hm	136D	0001	0000	0205	0208	00003388	d--lr-xr-xr-x	---F	/home/hm/
exceptions	13ED	0001	0000	0205	0208	00003388	d--lrwxr-xr-x	---F	/home/hm/exceptions/

```
typedef struct {
    unsigned __int32 fileno; // iFS, соль, iFile
    unsigned __int16 mode; // Режим доступа
    unsigned __int16 uid; // User ID владельца
    unsigned __int16 gid; // Group ID владельца
    unsigned __int16 salt; // Еще 16 бит соли
    char name[12]; // Имя файла
} T_MFS_Folder_Record; // 24 байта
```

Дамп директории home/policy/pwdmgr/

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

Биты	Состав поля fileno
11..0	Индекс файла iFile (0..4095)
27..12	16 бит соли
31..28	ID файловой системы (всегда 1)

Биты	Состав поля mode
8..0	rwxrwxrwx Unix-like права
9	I Integrity (целостность)
10	E Encryption (шифрование)
11	A Anti-Replay
13	N Использовать Non-Intel ключи
15..14	d Тип (0:файл,1:директория)

Если бит **I** установлен, к файлу добавляется 52 байта структуры `T_FileSecurity`

Целостность обязательно контролируется для:

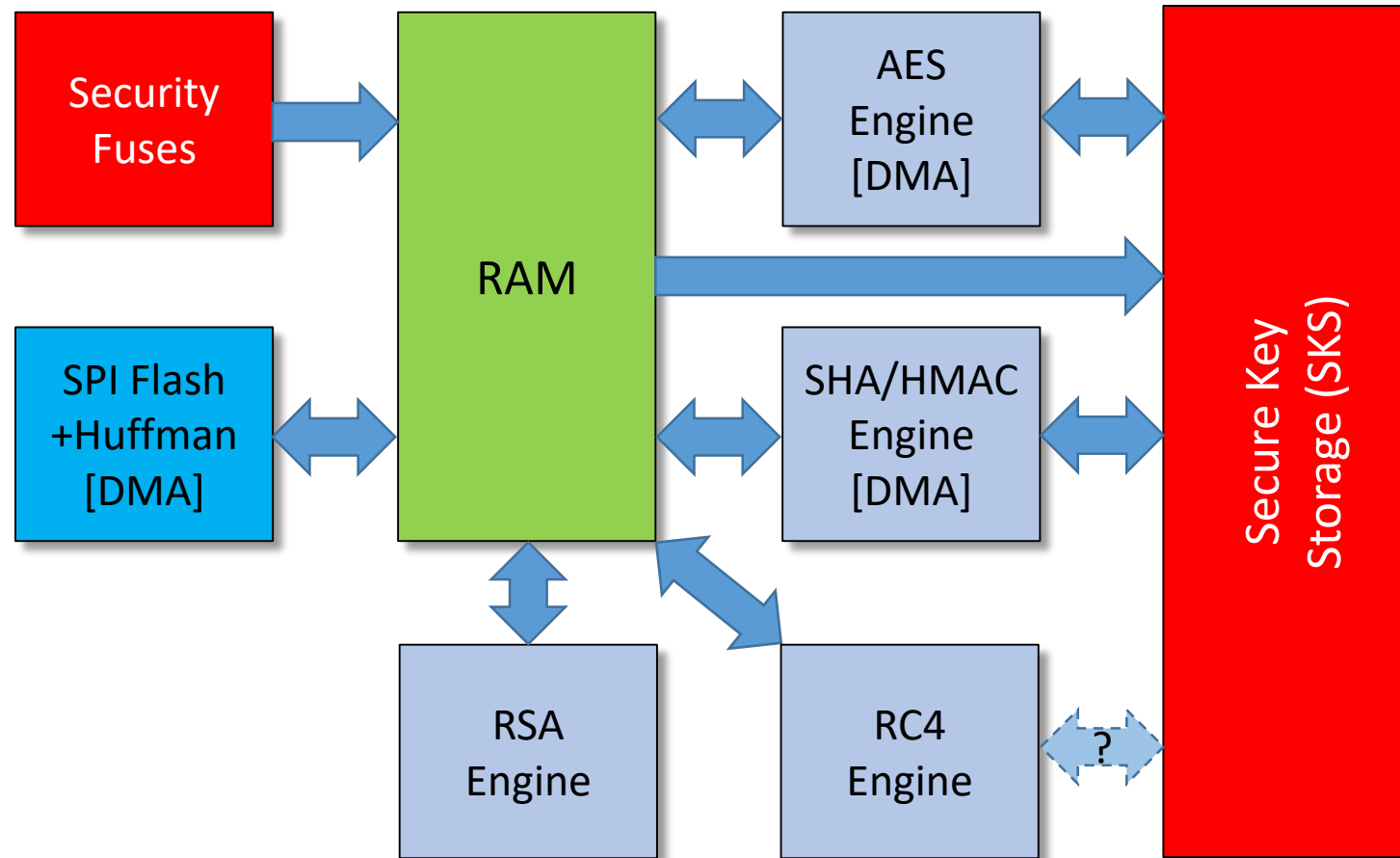
- AR-таблиц (`iFile == 2, 3`)
- Директории `/home/` (`iFile == 8`)

```
typedef struct {
    unsigned __int8 hmac[32]; // Значение HMAC
    unsigned __int32 antiReplay:2; // Anti-Replay
    unsigned __int32 encryption:1; // Encryption
    unsigned __int32 unk7:7;
    unsigned __int32 iAR:10; // Индекс в AR table
    unsigned __int32 unk12:12;
    union {
        struct ar { // Данные Anti-Replay
            unsigned __int32 rnd; // соль
            unsigned __int32 ctr; // счетчик
        };
        unsigned __int8 nonce[16]; // AES-CTR nonce
    };
} T_FileSecurity; // 52 байта
```

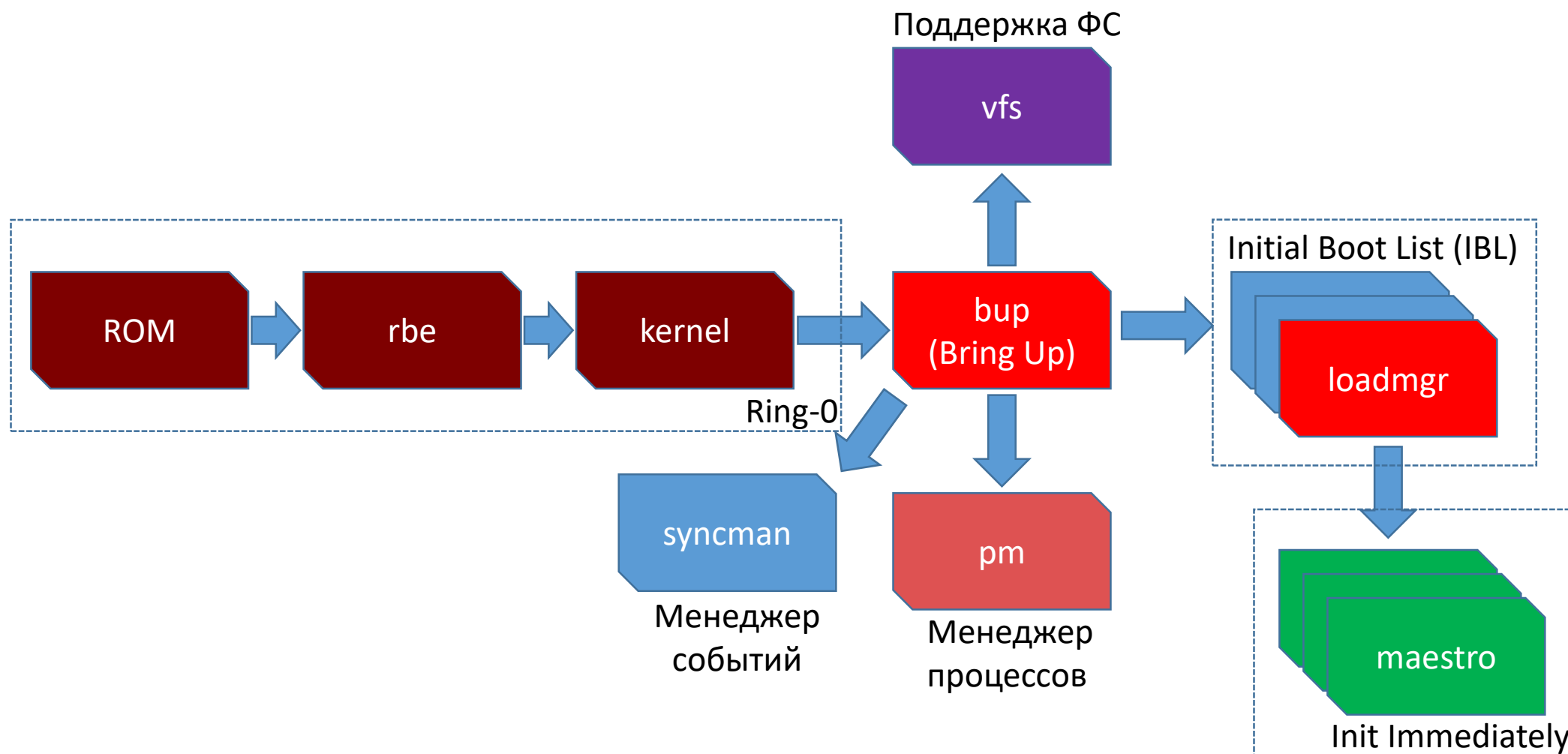
НМАС покрывает данные файла, структуру `T_FileSecurity` (с обнуленным полем `hmac`), а также поля `fileno` и `salt` (из директории)

Дополнительная информация

Особенности применения криптографии в МЕ 11



- Отображены в адресное пространство ME
- Доступны из режима ядра (ROM, rbe, kernel)
- Доступны приложениям ME, если в их манифесте прописаны соответствующие разрешения (доступ к RSA/AES/HMAC/SKS только у модулей buv и crypto)



- Содержит разделы BIOS/UEFI, GbE, ME, ...
- Хранит исполняемый код и настройки
- Может отображаться в память
- Имеет встроенный Huffman Decompressor (для кода модулей ME)

- Слоты 1..11 для 128-битовых, 12..21 для 256-битовых ключей
- Ключ может быть задан явно или взят из результата AES/HMAC
- Сохраненный ключ не может быть извлечен (только использован в связке с AES/HMAC)
- Существуют политики ключей (результат AES Encrypt можно положить в память, результат AES Decrypt – только в SKS)

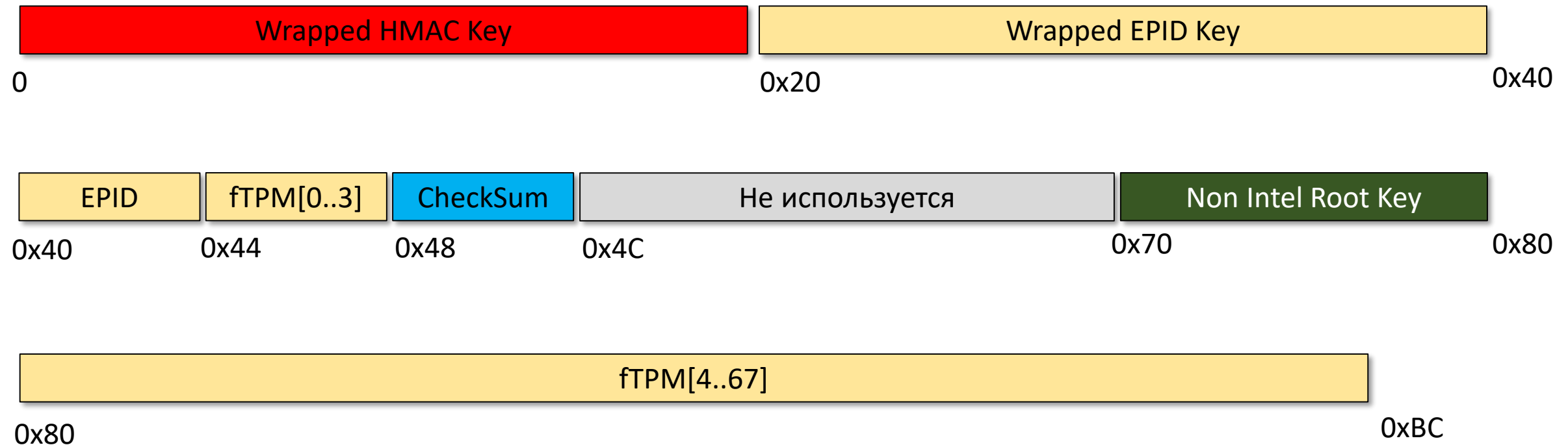
- Поддерживаются ключи размером 128 и 256 бит
- Поддерживаются режимы ECB, CBC, CTR
- Ключ шифрования может быть задан явно или взят из SKS
- Данные могут передаваться явно или через DMA

- Умеет вычислять SHA-1, SHA-256, SHA-384, SHA-512
- Поддерживаются ключи HMAC размером 128 и 256 бит
- Ключ HMAC может быть задан явно или взят из SKS
- Данные могут передаваться явно или через DMA
- Может работать в связке с AES

- Умеет выполнять модульное экспоненцирование
- Применяется в ROM (и не только) для проверки цифровой подписи
- Вероятно, используется приложениями ME

- Точно присутствует, но не исследовался нами
- Не используется для обеспечения безопасности собственно ME
- Вероятно, используется приложениями ME (для поддержки протоколов WiFi, SSL и т.п.)

- Инициализируются в процессе производства
- Не могут быть перезаписаны
- Доступны для чтения (после сброса платформы) только заданное число раз (обычно 1)
- Частично блокируются в случае активации JTAG



В обеспечение безопасности ФС вовлечено до 10 ключей

Intel Integrity	Non-Intel Integrity
Нынешние ключи (для текущего SVN)	
Intel Confidentiality	Non-Intel Confidentiality

Intel Integrity	Non-Intel Integrity
Предыдущие* ключи (могут отсутствовать)	
Intel Confidentiality	Non-Intel Confidentiality

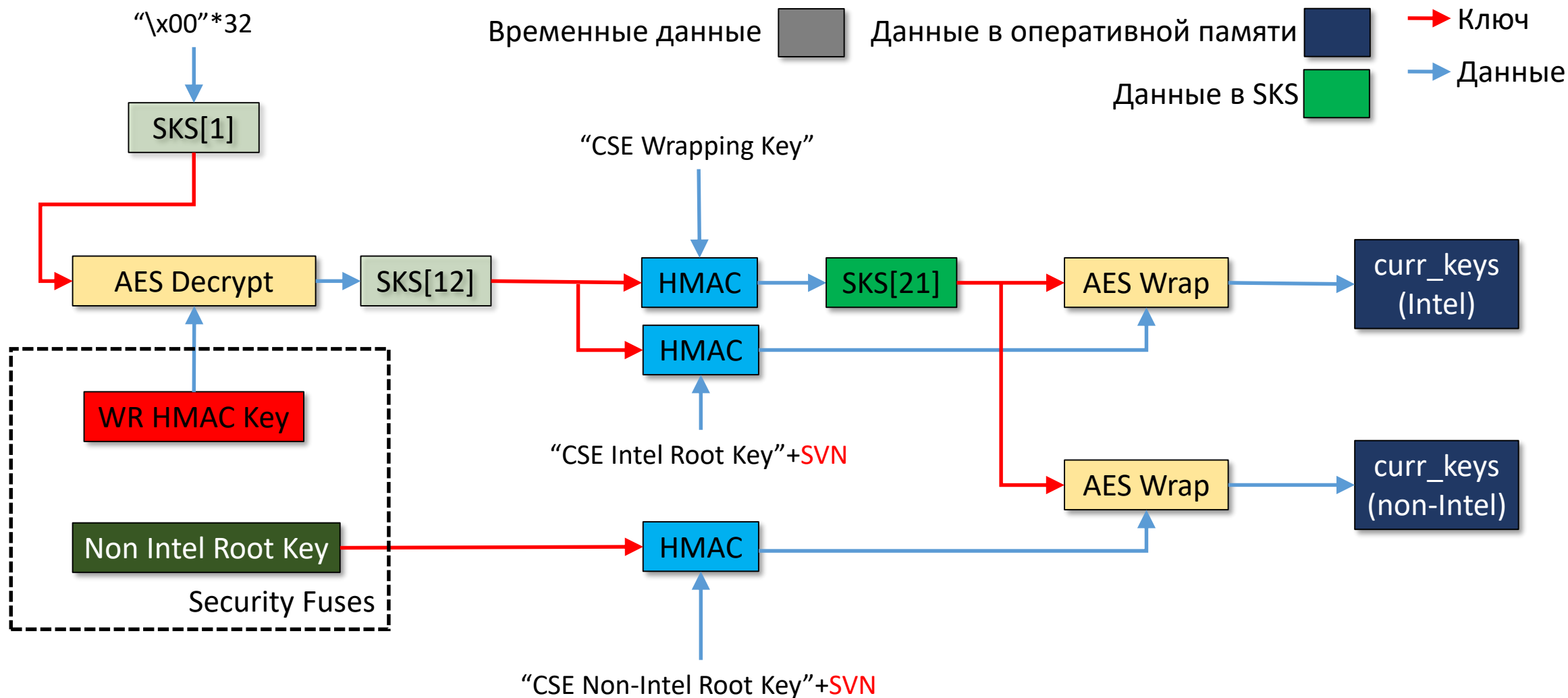
RPMC HMAC #0	RPMC HMAC #1
-----------------	-----------------

*Предыдущие ключи вычисляются если $SVN > 1$ и раздел PSVN содержит валидные данные. Эти ключи используются для миграции файлов, созданных до обновления значения SVN.

Replay-Protected Monotonic Counter (RPMC)
может быть реализован в микросхеме SPI Flash

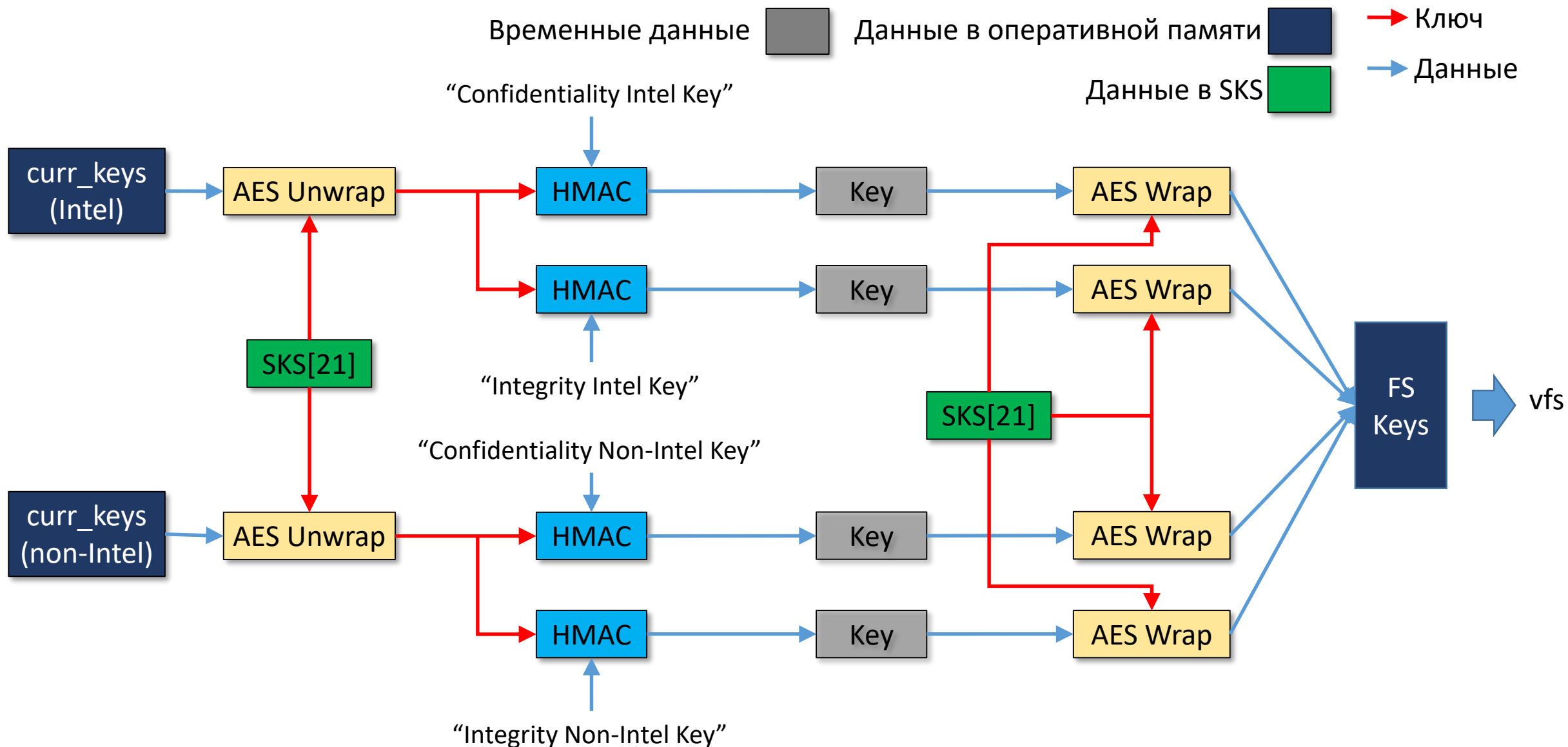
Генерация ключей файловой системы (ROM)

POSITIVE TECHNOLOGIES



Генерация ключей файловой системы (bup)

POSITIVE TECHNOLOGIES



Ключи для конфиденциальности/целостности данных в MFS	Intel	Non-Intel
Никогда ни не хранятся на Flash	Да	Да
Хранятся в памяти только «обернутые» на SKS-ключе #21	Да	Да
Не могут быть «развернуты» в память (только в SKS)	Да	Да
Зависят от однобайтового значения SVN	Да	Да
Зависят от данных, получаемых из устройства GEN (Security Fuses)	Да	Да
ROM очищает данные, прочитанные из GEN, перед запуском rbe	Да	Да
ROM блокирует доступ к устройству GEN перед запуском rbe	Да	Да
Секрет в устройстве GEN недоступен под JTAG	Да	Нет

Примечание: основной объем информации в MFS защищается через «Non-Intel keys»
«Intel keys» используются только модулями `sigma`, `ptt`, `dal_ivm`, `mca`

iFS	Имя	Описание
0	root	Определена в <code>vfs</code> . Вмещает до 1024 записей. Изначально содержит <code>/</code> , <code>/dev/</code> , <code>/etc/</code> , <code>/etc/rc</code> , <code>/temp/</code>
1	home	Для файлов из MFS, поддерживает Confidentiality/Integrity
2	bin	Отображает модули из Code Partition Directory (\$CPD)
3	susram	Определена в <code>bup</code> и <code>vfs</code> . Занимает 3072 байта NV Suspend RAM
4	fpf	Определена в <code>fpf</code> . Отсутствует в SPS (Server Platform Services)
5	dev	Содержит устройства из Special File Producer metadata extension
6	umafs	Никогда не видел, чтобы кто-то использовал...



Thank You!

POSITIVE TECHNOLOGIES

ptsecurity.com