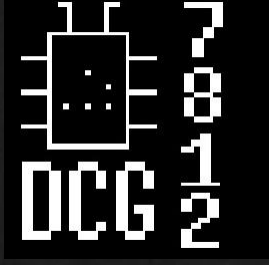


Safeguarding rootkits:
Intel Boot Guard

Part 2

Alexander Ermolov

Defcon Russia meetup #29

#disclaimer

1. No motherboards were harmed

2. The Intel Boot Guard implementation details given here is a result of a reverse engineering process, so it might contain some inaccuracy compared to the Intel Boot Guard specification (which is not public)

#whoami

Security researcher at  Digital Security

https://habrahabr.ru/users/flothrone/

a.ermolov@dsec.ru

flothrone@gmail.com

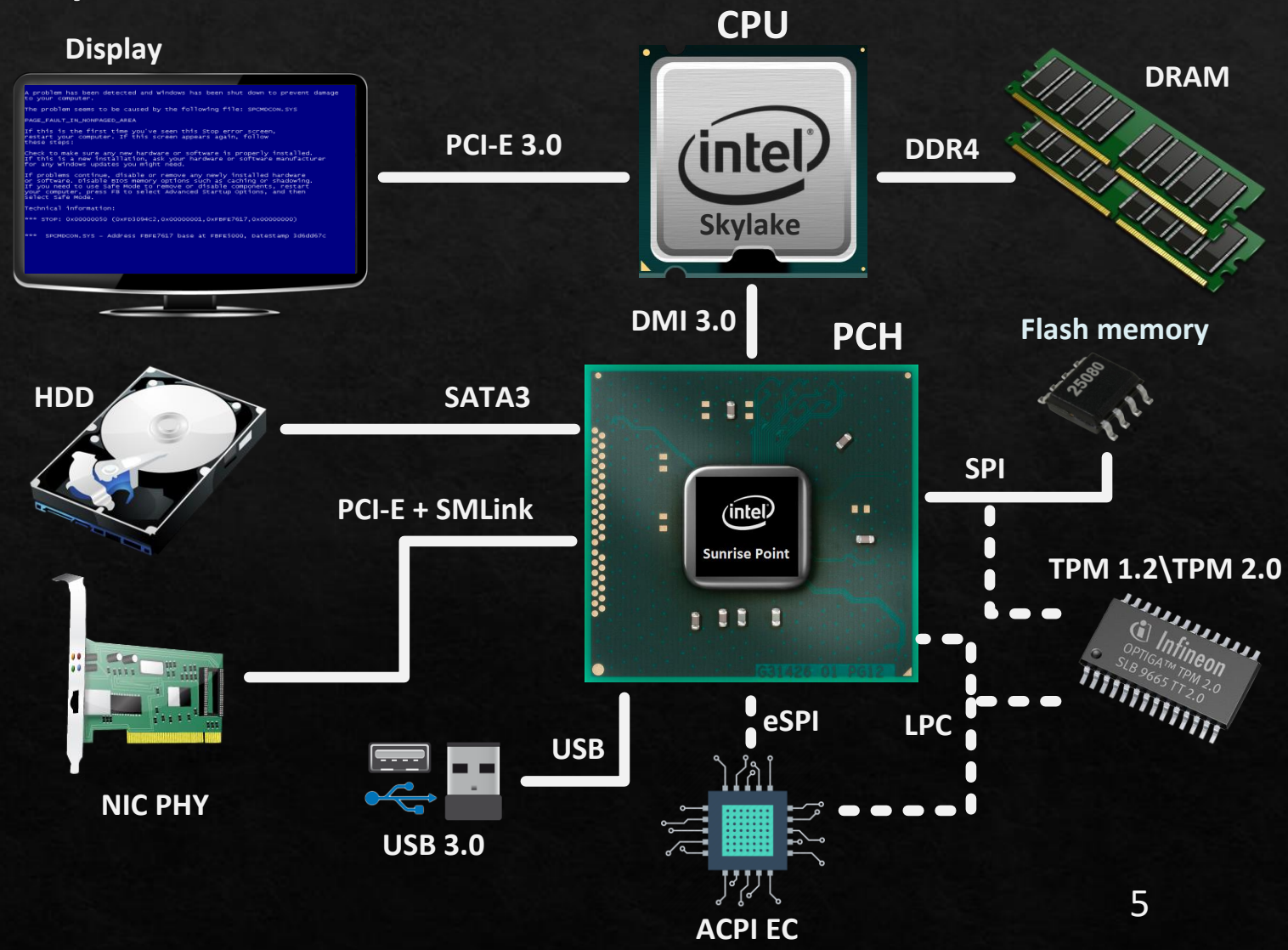
Intel x86 platform firmware

Desktop (laptop) system overview

Execution environments:

- CPU
- Chipset
- ACPI EC

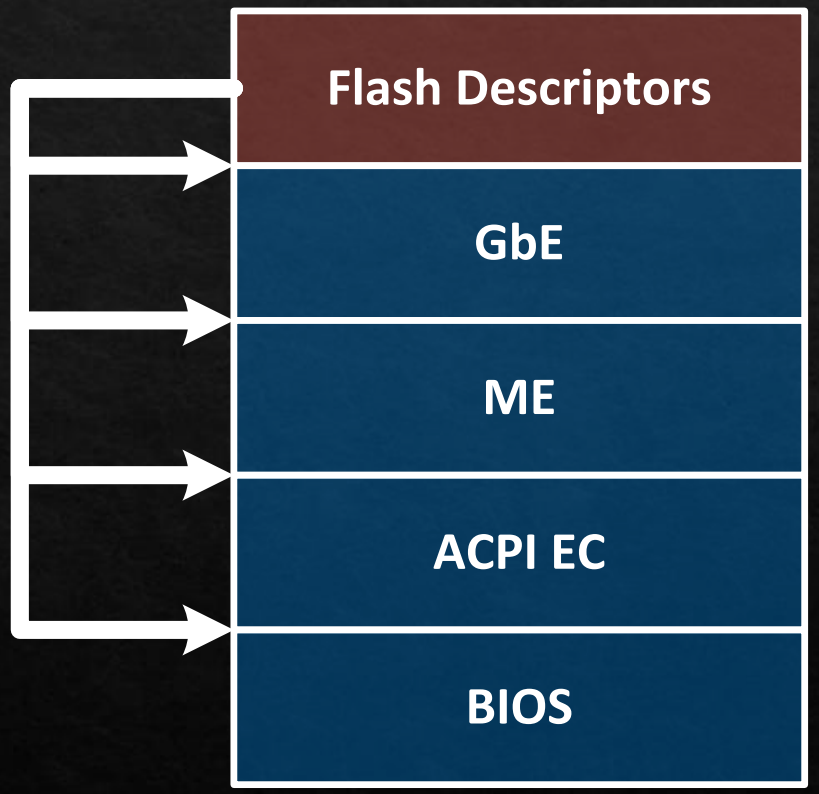
The main part of platform firmware is stored on SPI flash memory



SPI flash memory

The platform firmware is divided into regions:

- Flash Descriptors
  - Pointers to other regions
  - Access permissions
  - …
- GbE configuration
- ME
- ACPI EC (since Skylake)
- BIOS



Intel CPU

Main execution environment (BIOS\OS)

Privilege levels:

Ring 3	User Mode
…	
Ring 0	Kernel Mode
Ring -1	Hypervisor Mode
Ring -2	System Management Mode (SMM)

Intel CPU

Root of Trust

- Microcode ROM (== Boot ROM ?)
- AES key for decrypting microcode updates
- Hash of an RSA public key which verifies the microcode updates
- Hash of an RSA public key which verifies other Intel blobs (e.g. ACMs…)

Intel ME

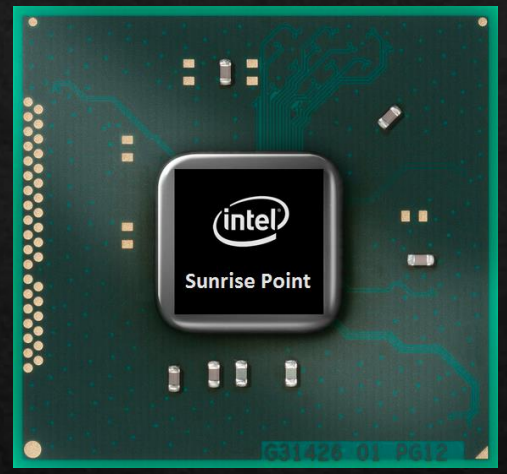
The chipset subsystem integrated into:

- Q-type chipsets since 960 series (2006 - 2009)

  Intel ME 2.x – 5.x

- All chipsets since 5 series (2010 - …)

  Intel ME 6.x – 11.x, TXE 1.x – 3.x, SPS 1.x – 4.x



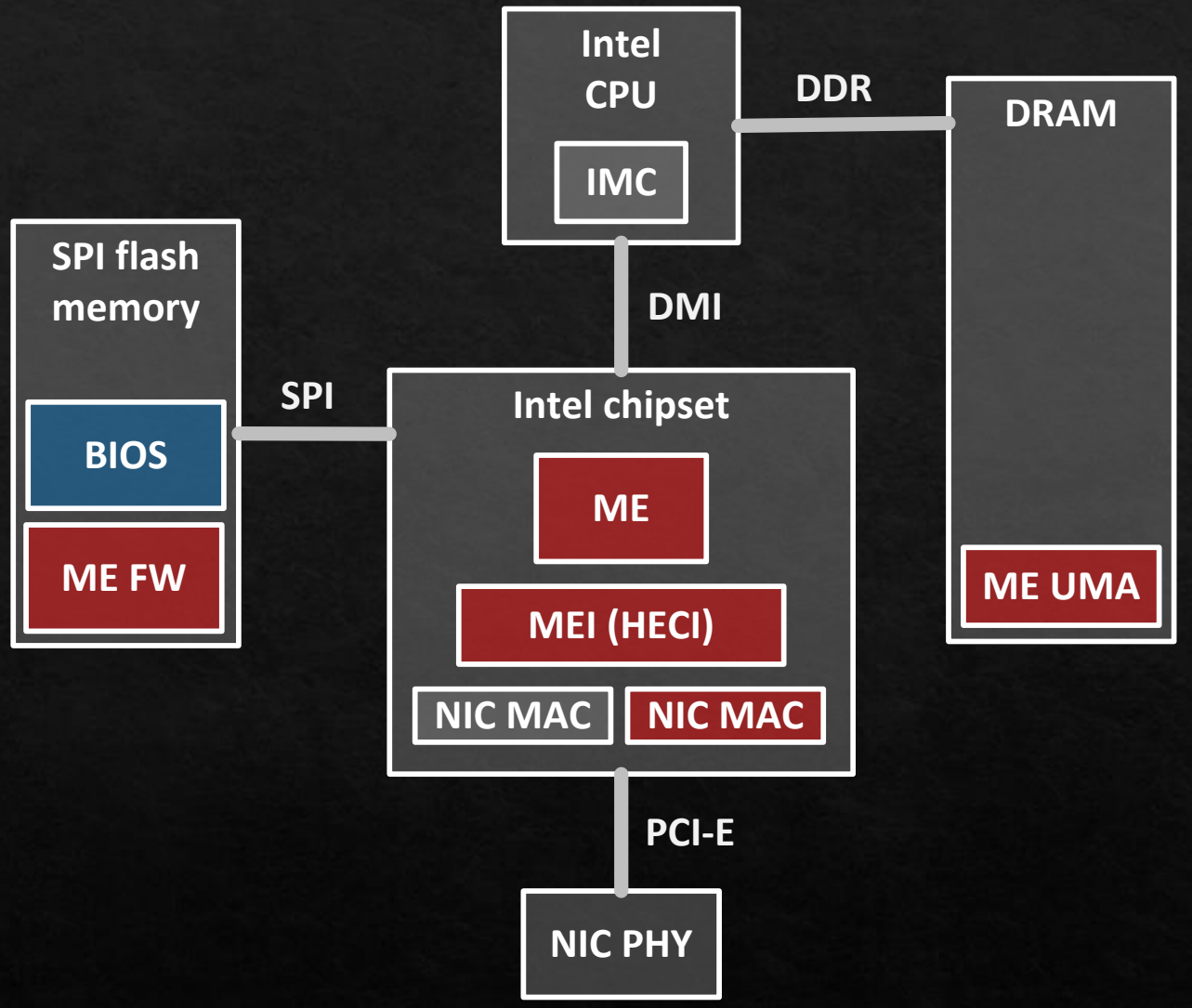
Platforms affected:

• Desktop, Laptop	Intel Management Engine (ME)
• Mobile	Intel Trusted Execution Engine (TXE) / Security Engine (SeC)
• Server	Intel Server Platform Services (SPS)

Intel ME

The most privileged and hidden execution environment (Ring -3):

- Hidden from CPU runtime memory in DRAM
- Full access to DRAM
- Working even when CPU is in S5 (system shutdown)
- Out-of-Band (OOB) access to network interface
- Runs firmware (based on RTOS ThreadX) from common SPI flash



Intel ME

Root of Trust

- ME ROM with the bootcode
- Hash of an RSA public key which verifies code partitions of ME FW
- AES key to store sensitive data
- Field Programmable Fuses (FPFs) to permanently store some configuration

Intel Integrated Sensor Hub (ISH)

Integrated in Intel SoC since ? Bay Trail ?

Seems to be truncated version of Intel ME:

- ROM with the bootcode and SRAM
- Has its own HECI
- Has a DMA engine (? shares some memory with ME ?)
- Runs firmware (ISHC partition of ME FW) from common SPI flash

Firmware can be developed and signed by Intel/OEM

Intel System Tool Kit (STK)

Intel provides these tools for OEMs for building system firmware images:

- Flash Image Tool
- Flash Programming Tool
- FWUpdate
- MEinfo
- MEmanuf
- …

BIOS protection mechanisms

Protection against modifications from software

- Physical protection
  - Hardware Write Protect jumper
- Map protection
  - Protected Range (PR) registers
- SMM
  - BLE (BIOS_WE) / SMM_BWP
- SMM over SMM
  - Intel BIOS Guard (PFAT)

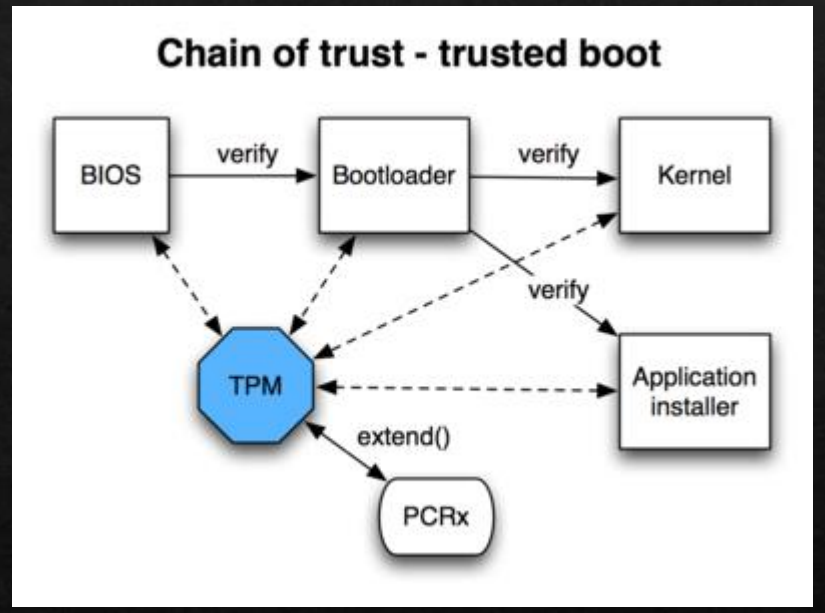
Though some vendors using a few of these (and not always implemented), but there are always many that don’t care…

Verification (trusted boot) mechanisms

- Secure Boot
- Hardware-assisted Secure Boot

  Bay Trail
- Intel Boot Guard

  Haswell / Braswell / Skylake / Apollo Lake / Kaby Lake …

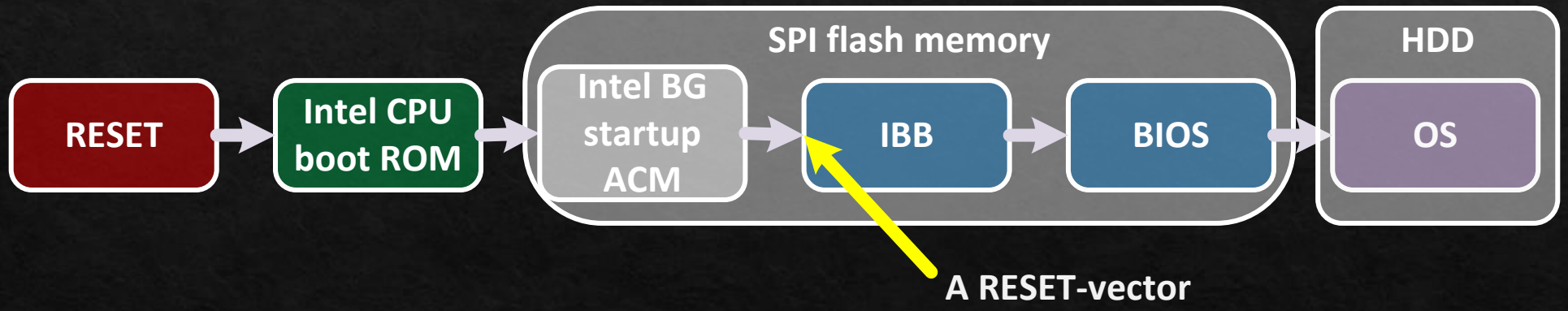


Intel Boot Guard 1.x*

* - not an official version number, this is how I order its versions

Intel Boot Guard (BG)

A hardware-based boot integrity protection available since Haswell



Operating modes:

- Measured Boot (MB)
- Verified Boot (VB)
- MB + VB

Intel BG. Measured Boot

Uses the Trusted Platform Module (TPM) Platform Configuration Registers (PCRs) to reflect boot components integrity

```
Measure (data):
   PCR = Hash(PCR | Hash(data))
```

Some sensitive data can be sealed (`TPM_Seal`) to the PCRs state

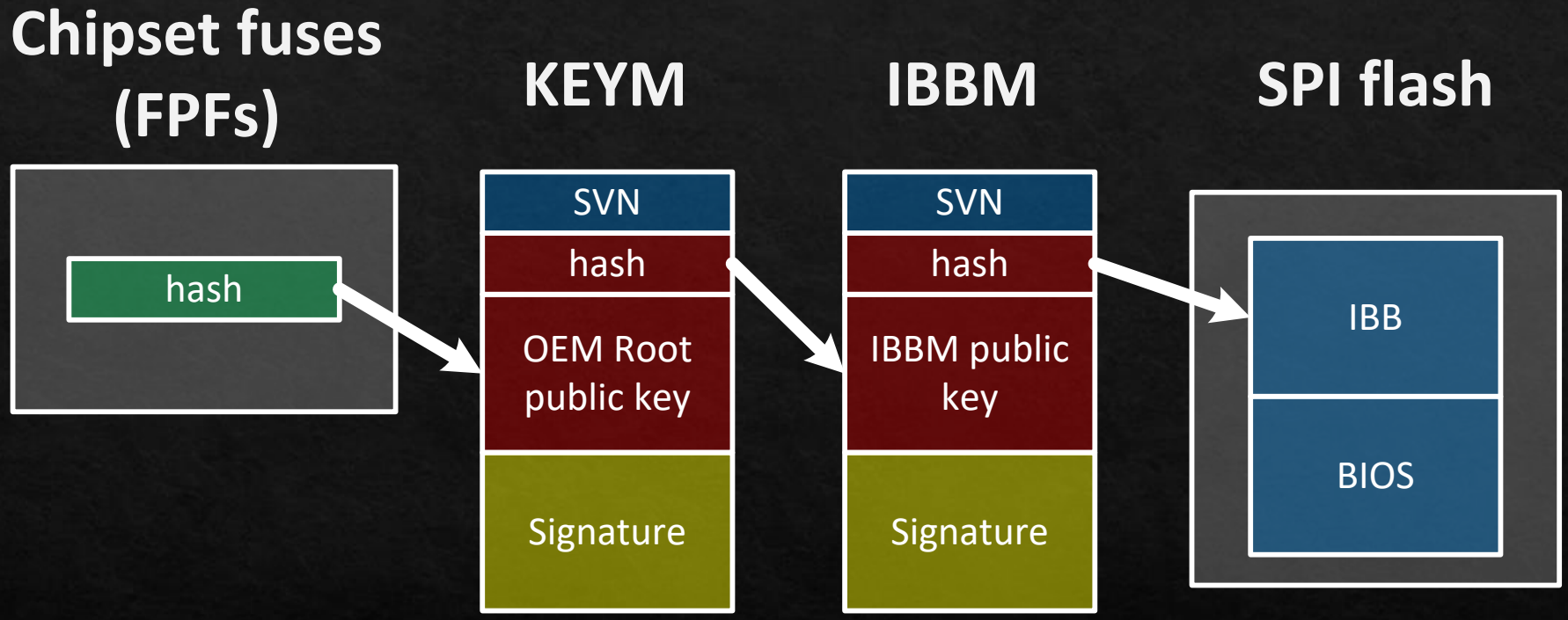
Intel BG. Verified Boot

Cryptographically verifies the integrity of boot components

Options, in case of a verification fail (enforcement policy):

- Do nothing
- Force an immediate shutdown
- Force a shutdown upon a timeout (e.g. 1 or 30 minutes)

Intel BG. Verified Boot

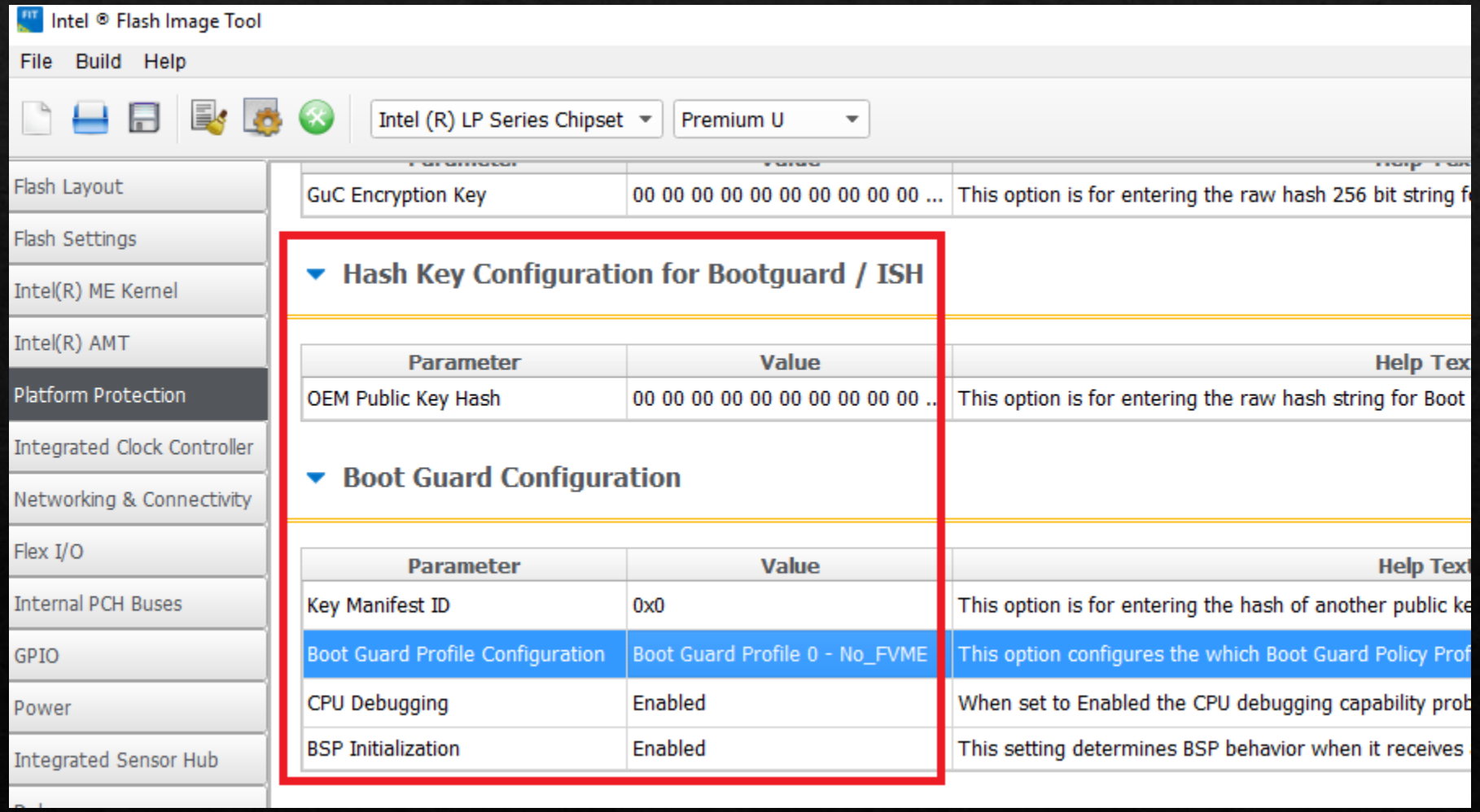


If the OEM Root private key is compromised, there is no way to replace/ revoke it (as long as it's hash is in permanent storage)

The unique IBBM public key can be used for different product lines

So in case of one IBBM private key is compromised, it affects only one product line until this key is replaced

Intel BG. Configuration



Intel BG. Configuration

```
typedef struct BG_PROFILE
{
    unsigned long Force_Boot_Guard_ACM : 1;
    unsigned long Verified_Boot : 1;
    unsigned long Measured_Boot : 1;
    unsigned long Protect_BIOS_Environment : 1;
    unsigned long Enforcement_Policy : 2;     // 00b - do nothing
                                              // 01b - shutdown with timeout
                                              // 11b - immediate shutdown
    unsigned long : 26;
};
```

Intel BG. Configuration

BG profiles:

- No_FVME Disabled
- VE VB, shutdown timeout
- VME VB + MB, shutdown timeout
- VM VB + MB, do nothing
- FVE VB, immediate shutdown
- FVME VB + MB, immediate shutdown

Intel BG. Configuration

The Intel BG configuration is created by OEM and permanently saved to Field Programmable Fuses (FPFs) - the hardware non-volatile storage inside Intel chipset (only Intel ME can program and read them)

FPFs fits perfect to store the configuration:

- Fuses can be one-time programmable
- Access only through Intel ME

Intel BG. Configuration procedure

Intel Flash Image Tool:

1) Prepare the system firmware image with configured ME NVARs (Intel BG configuration) that are to be committed to FPFs

Intel Flash Programming Tool:

2) Write the new image to the SPI flash memory

3) Close the manufacturing mode (this will commit an appropriate ME NVARs to FPFs, lock down the SPI flash regions and issue a global RESET)

Researched systems

Let’s take a closer look on Intel BG implementation…

- Gigabyte GA-H170-D3H — BG support present
- Gigabyte GA-Q170-D3H — BG support present
- Gigabyte GA-B150-HD3 — BG support present
- MSI H170A Gaming Pro — BG support not present
- Lenovo ThinkPad 460 — BG support present, BG enabled
- Lenovo Yoga 2 Pro — BG support not present
- Lenovo U330p — BG support not present

Intel CPU boot ROM

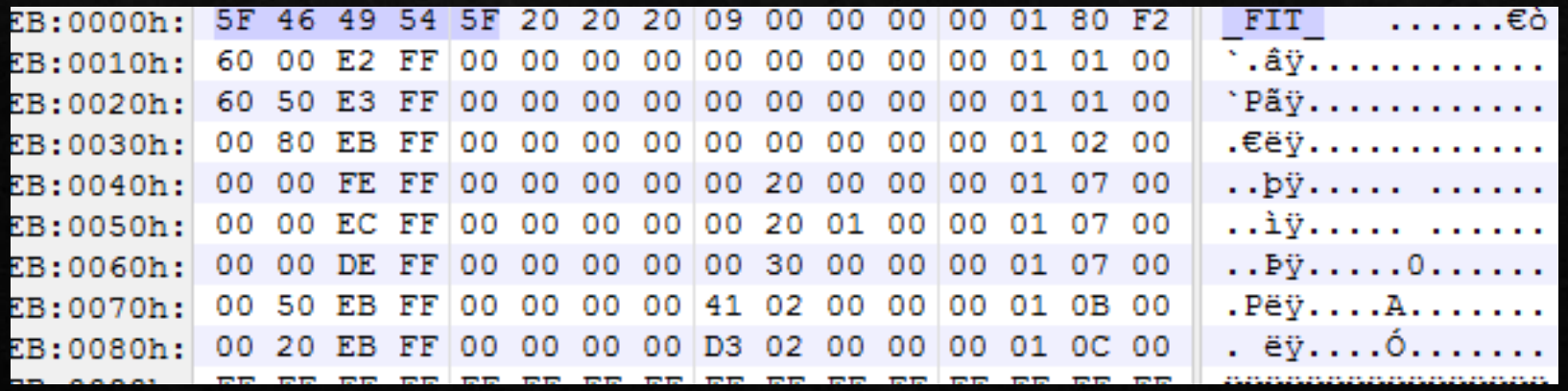
No image of it for researching, but some docs mention that it does:

1) Find the Firmware Interface Table (FIT)

FIT base address is located at 0xFFFFFFC0

2) Find Intel BG startup Authenticated Code Module (ACM), verify, load and execute it

FIT contains the base address of Intel BG startup ACM



EB:0000h:	5F	46	49	54	5F	20	20	20	09	00	00	00	00	01	80	F2	_FIT_ ......€ò
EB:0010h:	60	00	E2	FF	00	00	00	00	00	00	00	00	00	01	01	00	`.âÿ..........
EB:0020h:	60	50	E3	FF	00	00	00	00	00	00	00	00	00	01	01	00	`Pãÿ..........
EB:0030h:	00	80	EB	FF	00	00	00	00	00	00	00	00	00	01	02	00	.€ëÿ..........
EB:0040h:	00	00	FE	FF	00	00	00	00	00	20	00	00	00	01	07	00	..þÿ..... .....
EB:0050h:	00	00	EC	FF	00	00	00	00	00	20	01	00	00	01	07	00	..ìÿ..... .....
EB:0060h:	00	00	DE	FF	00	00	00	00	00	30	00	00	00	01	07	00	..Þÿ.....0.....
EB:0070h:	00	50	EB	FF	00	00	00	00	41	02	00	00	00	01	0B	00	.Pëÿ....A.....
EB:0080h:	00	20	EB	FF	00	00	00	00	D3	02	00	00	00	01	0C	00	. ëÿ....Ó.....

Intel CPU boot ROM

The FIT is a table of few entries and the first entry is a FIT header

```
typedef struct FIT_HEADER
{
   char           Tag[8];      // '_FIT_   '
   unsigned long  NumEntries; // including FIT header entry
   unsigned short Version;     // 1.0
   unsigned char  EntryType;   // 0
   unsigned char  Checksum;
};
```

Intel CPU boot ROM

Other FIT entries have the same format

They describes Intel blobs that are to be parsed\executed before the execution of BIOS, hence before the Legacy RESET-vector (0xFFFFFFF0)

```
typedef struct FIT_ENTRY
{
    unsigned long  BaseAddress;
    unsigned long  : 32;
    unsigned long  Size;
    unsigned short Version;     // 1.0
    unsigned char  EntryType;
    unsigned char  Checksum;
};
```

Intel CPU boot ROM

```
enum FIT_ENTRY_TYPES
{
    FIT_HEADER = 0,
    MICROCODE_UPDATE,
    BG_ACM,
    BIOS_INIT = 7,
    TPM_POLICY,
    BIOS_POLICY,
    TXT_POLICY,
    BG_KEYM,
    BG_IBBM
};
```

Intel BG startup ACM

```
00003BB1 Start          proc near
00003BB1                mov     ax, ds
00003BB4                mov     ss, ax
00003BB7                mov     es, ax
00003BBA                mov     fs, ax
00003BBD                mov     gs, ax
00003BC0                mov     esp, ebp
00003BC2                add     esp, 1000h
00003BC8                mov     eax, ebp
00003BCA                add     eax, 4C8h
00003BCF                lidt    fword ptr [eax]
00003BD2                push    ebp
00003BD3                call    BootGuard__
00003BD8                mov     ebx, eax
00003BDA                mov     edx, 0
00003BDF                mov     eax, 3
00003BE4                getsec
```

```
00003912 BootGuard__    proc near               ; CODE XR
00003912
00003912 var_10         = dword ptr -10h
00003912 var_C          = dword ptr -0Ch
00003912 var_8          = dword ptr -8
00003912 var_4          = dword ptr -4
00003912 arg_0          = dword ptr  8
00003912
                        push    ebp
                        mov     ebp, esp
                        sub     esp, 10h
                        push    ebx
                        mov     ebx, [ebp+arg_0]
                        push    esi
                        push    edi
                        xor     eax, eax
                        lea     esi, [ebx+6000h]
                        push    ebx
                        push    esi
                        mov     [ebp+var_10], 0FFF0h
                        mov     [ebp+var_C], eax
                        mov     [ebp+var_8], eax
                        mov     [ebp+var_4], eax
                        call    PlatformInit__
                        mov     edi, eax
                        pop     ecx
                        pop     ecx
00003940                test    edi, edi
00003941                jnz     loc_3A3E
00003943                movzx   eax, word ptr [esi+1F0Eh]
00003949                test    al, 3
00003950                jz      loc_3ADA
00003952                push    esi             ; int
00003958                call    GetBootGuardData__
00003959                mov     edi, eax
0000395E                pop     ecx
00003960                test    edi, edi
00003961                jnz     loc_3A3E
00003963                lea     eax, [ebp+var_C]
00003969                push    eax
0000396C                push    esi
0000396D                call    BootGuardInit__
0000396E                mov     edi, eax
00003973
```

```
00004345 BootGuardInit__ proc near              ; CODE XR
00004345
00004345 arg_0          = dword ptr  4
00004345 arg_4          = dword ptr  8
00004345
00004345                push    edi
00004346                mov     edi, [esp+4+arg_0]
0000434A                call    KeyM__
0000434F                test    eax, eax
00004351                jnz     short loc_4384
00004353                mov     eax, edi
00004355                call    IbbM__
0000435A                test    eax, eax
0000435C                jnz     short loc_4384
0000435E                mov     edx, [edi+1F28h]
00004364                mov     ecx, [esp+4+arg_4]
00004368                add     [ecx], edx
0000436A                mov     edx, [ecx]
0000436C                add     edx, [edi+1F30h]
00004372                push    esi
00004373                mov     [ecx], edx
00004375                movzx   esi, word ptr [edi+15BEh]
0000437C                shl     esi, 0Ch
0000437F                add     esi, edx
00004381                mov     [ecx], esi
00004383                pop     esi
00004384
00004384 loc_4384:                              ; CODE XR
00004384                                        ; BootGua
00004384                pop     edi
00004385                retn
00004385 BootGuardInit__ endp
```

Intel BG startup ACM

Parse FIT:

1) Retrieve hash of OEM Root public key and Boot Policies from FPFs (through Intel ME)

2) Locate Key Manifest (KEYM) and verify it

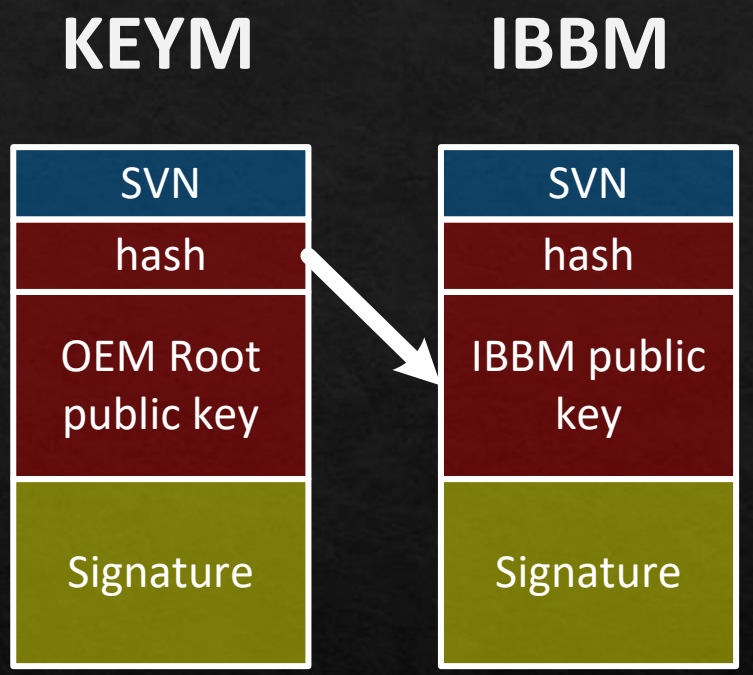
3) Locate IBB Manifest (IBBM) and verify it

Intel BG startup ACM

```
enum FIT_ENTRY_TYPES
{
    FIT_HEADER = 0,
    MICROCODE_UPDATE,
    BG_ACM,
    BIOS_INIT = 7,
    TPM_POLICY,
    BIOS_POLICY,
    TXT_POLICY,
    BG_KEYM,
    BG_IBBM
};
```

Intel BG startup ACM

```
typedef struct KEY_MANIFEST
{
    char           Tag[8];          // '__KEYM__'
    unsigned char  : 8;             // 10h
    unsigned char  : 8;             // 10h
    unsigned char  : 8;             // 0
    unsigned char  : 8;             // 1
    unsigned short : 16;            // 0Bh
    unsigned short : 16;            // 20h == hash size?
    unsigned char  IbbmKeyHash[32]; // SHA256 of an IBBM public key
    BG_RSA_ENTRY   OemRootKey;
};
```



Intel BG startup ACM

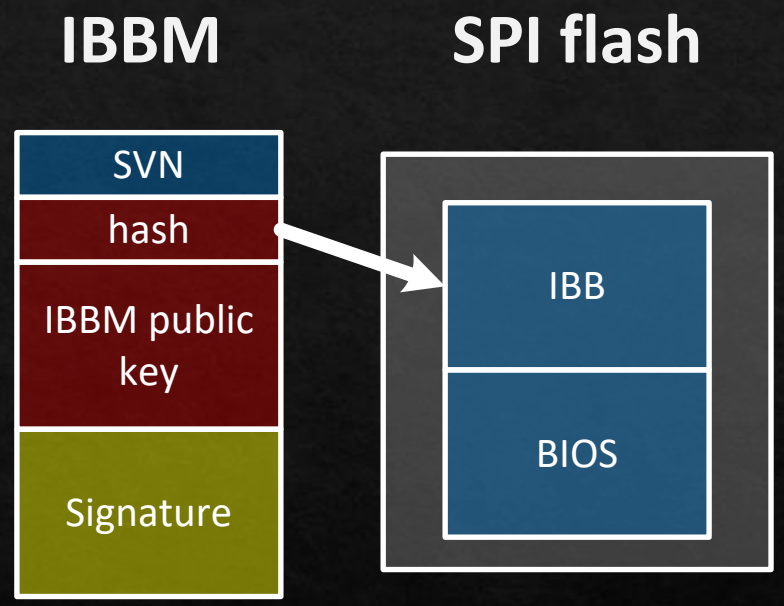
```
typedef struct BG_RSA_ENTRY
{
    unsigned char  : 8;              // 10h
    unsigned short : 16;             // 1
    unsigned char  : 8;              // 10h
    unsigned short RsaPubKeySize;    // 800h
    unsigned long  RsaPubExp;
    unsigned char  RsaPubKey[256];
    unsigned short : 16;             // 14
    unsigned char  : 8;              // 10h
    unsigned short RsaSigSize;       // 800h
    unsigned short : 16;             // 0Bh
    unsigned char  RsaSig[256];
};
```

Intel BG startup ACM

```
typedef struct IBB_MANIFEST
{
    ACBP Acbp;              // Boot policies

    IBBS Ibbs;              // IBB description
    IBB_DESCRIPTORS[];

    PMSG Pmsg;              // IBBM signature
};
```



Intel BG startup ACM

```
typedef struct ACBP
{
    char           Tag[8];          // '__ACBP__'
    unsigned char  : 8;             // 10h
    unsigned char  : 8;             // 1
    unsigned char  : 8;             // 10h
    unsigned char  : 8;             // 0
    unsigned short : 16;            // x & F0h = 0
    unsigned short : 16;            // 0 < x <= 400h
};
```

Intel BG startup ACM

```
typedef struct IBBS
{
    char           Tag[8];              // '__IBBS__'
    unsigned char  : 8;                 // 10h
    unsigned char  : 8;                 // 0
    unsigned char  : 8;                 // 0
    unsigned char  : 8;                 // x <= 0Fh
    unsigned long  : 32;                // x & FFFFFFF8h = 0
    unsigned long  Unknown[20];
    unsigned short : 16;                // 0Bh
    unsigned short : 16;                // 20h == hash size ?
    unsigned char  IbbHash[32];         // SHA256 of an IBB
    unsigned char  NumIbbDescriptors;
};
```

Intel BG startup ACM

Initial Boot Block (IBB) content is described in IBB_DESCRIPTORS

```
typedef struct IBB_DESCRIPTOR
{
    unsigned long  : 32;
    unsigned long  BaseAddress;
    unsigned long  Size;
};
```

So the concatenation of blocks (usually all SEC/PEI modules in UEFI image) pointed by IBB descriptors forms the IBB

Intel BG startup ACM

```
typedef struct PMSG
{
    char            Tag[8];             // '__PMSG__'
    unsigned char   : 8;                // 10h
    BG_RSA_ENTRY    IbbKey;
};
```

IBB

Hence, the SEC/PEI code is verified before the CPU starts executing from the RESET vector (FFFFFFF0h)

Then the BootGuard supporting code in PEI must verify the DXE volumes

Such PEI module is developed by OEM, e.g.:

- Lenovo

  LenovoVerifiedBootPei {B9F2AC77-54C7-4075-B42E-C36325A9468D}

- Gigabyte

  BootGuardPei {B41956E1-7CA2-42DB-9562-168389F0F066}

LenovoVerifiedBootPei

```
if (EFI_PEI_SERVICES->GetBootMode() != BOOT_ON_S3_RESUME)
{
   if (!FindHashTable())
      return EFI_NOT_FOUND;

   if (!VerifyDxe())
      return EFI_SECURITY_VIOLATION;
}
```

LenovoVerifiedBootPei

Hash table PEI module {389CC6F2-1EA8-467B-AB8A-78E769AE2A15}

```
typedef struct HASH_TABLE
{
    char          Tag[8];            // '$HASHTBL'
    unsigned long NumDxeDescriptors;

    DXE_DESCRIPTORS[];
};

typedef struct DXE_DESCRIPTOR
{
    unsigned char BlockHash[32];     // SHA256
    unsigned long Offset;
    unsigned long Size;
};
```

BootGuardPei

```
int bootMode = EFI_PEI_SERVICES->GetBootMode();

if (bootMode != BOOT_ON_S3_RESUME &&
    bootMode != BOOT_ON_FLASH_UPDATE &&
    bootMode != BOOT_IN_RECOVERY_MODE)
{
    HOB* h = CreateHob();

    if (!FindHashTable())
        return EFI_NOT_FOUND;

    WriteHob(&h, VerifyDxe());

    return h;
}
```

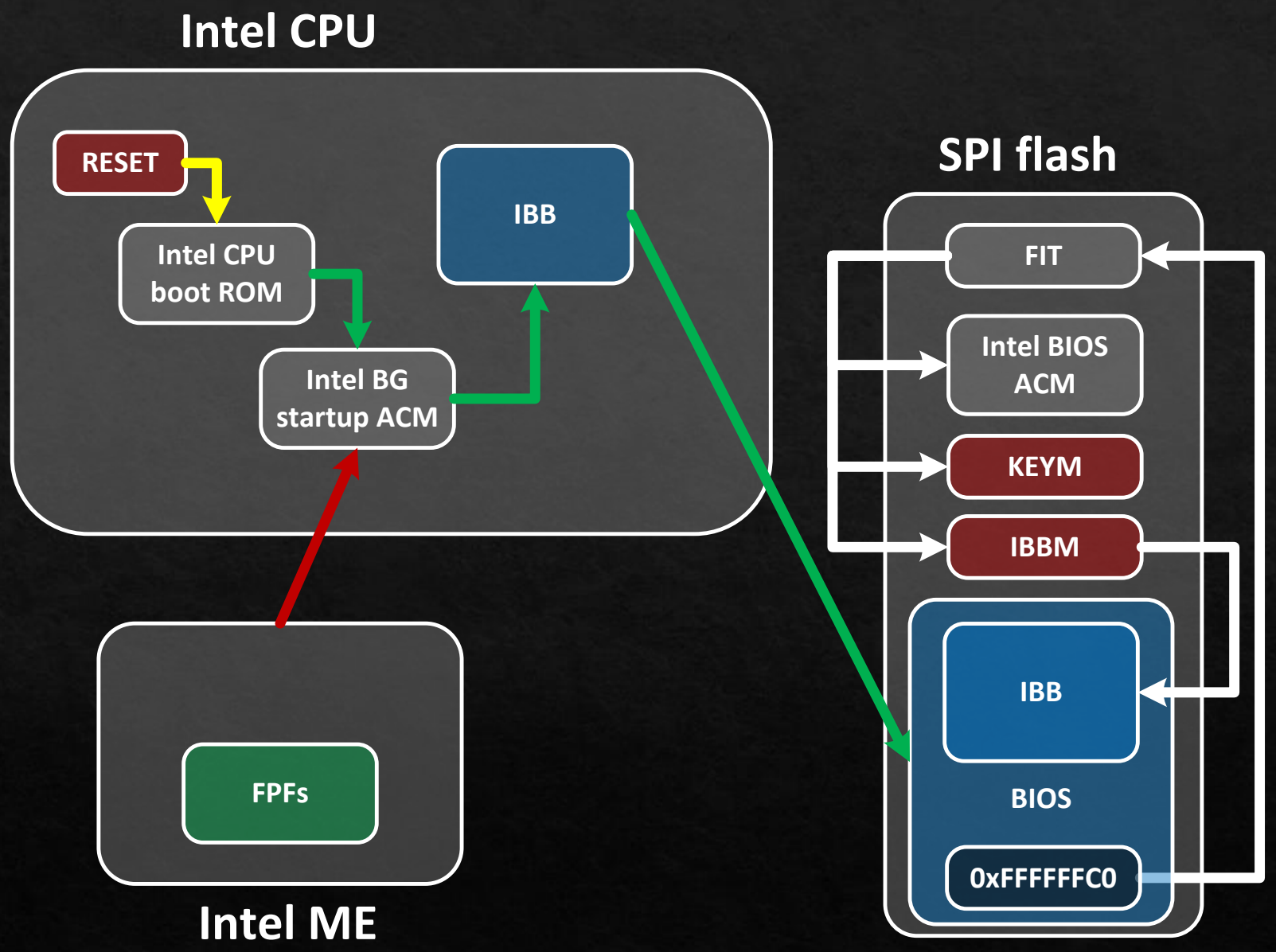
BootGuardPei

Hash table PEI module {389CC6F2-1EA8-467B-AB8A-78E769AE2A15}

```
typedef HASH_TABLE DXE_DESCRIPTORS[];

typedef struct DXE_DESCRIPTOR
{
   unsigned char BlockHash[32];      // SHA256
   unsigned long BaseAddress;
   unsigned long Size;
};
```

Overview


Intel CPU
RESET
Intel CPU boot ROM
Intel BG startup ACM
IBB
FPFs
Intel ME
SPI flash
FIT
Intel BIOS ACM
KEYM
IBBM
IBB
BIOS
0xFFFFFFC0

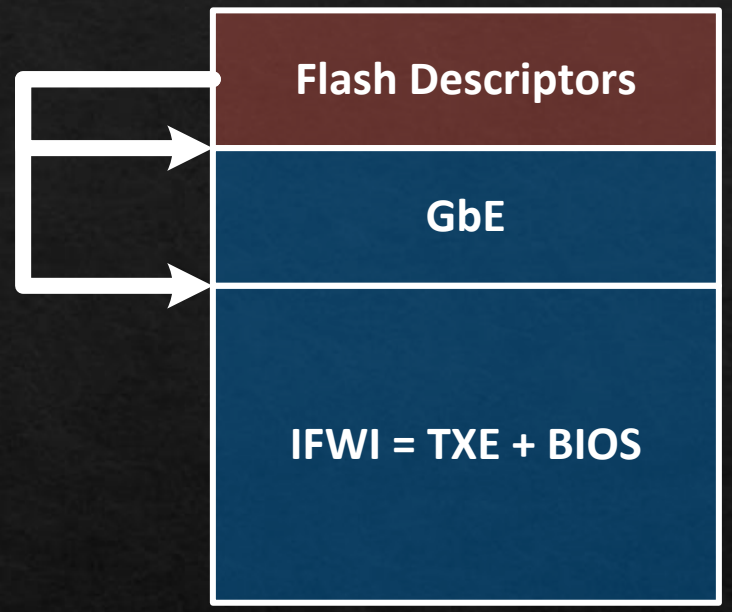
Intel Boot Guard 2.x*

* - not an official version number, this is how I order its versions

Architectural changes

Available only in Intel SoCs since Apollo Lake

Implementation found on ASRock J4205-ITX



First difference:

- BIOS and ME/TXE region have become a single region of SPI flash (IFWI)
- No FIT, KEYM, IBBM for Intel BG found
- PMC (Power Management Controller) - a new ARC core in the chipset

IFWI region

0000 2000h	SMIP	a kind of a platform configuration, signed by OEM
0000 6000h	RBEP	an x86 code for ME, signed by Intel
0001 0000h	PMCP	an ARC code for PMC, signed by Intel
0002 0000h	FTPR	an x86 code for ME, signed by Intel
0007 B000h	UCOD	microcode update for CPU, signed by Intel
0008 0000h	IBBP	SEC/PEI code of the BIOS, signed by OEM
0021 8000h	ISHC	an x86 code for ISH, signed by OEM
0025 8000h	NFTP	an x86 code for ME, signed by Intel
0036 1000h	IUNP	I don't know what is this
0038 1000h	OBBP	DXE code of the BIOS, not signed

Intel Boot Guard

Analyzing the TXE code shows that:

- The TXE starts first and controls the reset state of CPU (does not release it before everything is ready)
- The TXE prepares initial address space for CPU (FIT, BG startup ACM, KEYM, IBBM …) in it’s SRAM which will be temporary shared to the CPU
- …

Safeguarding rootkits

The issue

One day I've found out that some systems have the SPI flash regions unlocked and the BootGuard configuration left undefined (nor enabled, nor disabled):

- All Gigabyte systems
- All MSI systems
- 21 Lenovo branded notebook machine types and 4 ThinkServer machine types
- …

That's because of the close manufacturing fuse was not set at the end of the manufacturing line.

Lenovo Statement

«Lenovo has released fixes for the affected products, which can be found at https://support.lenovo.com/solutions/LEN_9903 or via our security advisory website, https://support.lenovo.com/product_security, and we have adjusted manufacturing processes, where necessary, to prevent reoccurrence of this issue in the future. We sincerely appreciate Mr. Ermolov's responsible disclosure and partnership in this matter.»

Intel Statement

«Intel’s guidance to our business partners is to close manufacturing mode at the end of production in order to maximize the security of the platform.»

Safeguarding rootkits

So any user could configure the Intel BG instead of OEM:

- Load into OS
- Modify BIOS
- Write proper BG configuration and verification entities (KEYM, IBBM) using Intel Flash Image Tool
- Set the closemnf fuse using the Intel Flash Programming Tool

This will permanently enable Intel BG on the system and will protect modified BIOS

DEMO

Safeguarding rootkits

The rootkit can be an SMM driver with the following capabilities:

1) Executed during OS

- Registers a SMI ISR and configure a timer to generate SMI events

2) Full (except ME UMA) access to CPU physical address space and complete isolation from OS

- SMRAM

3) An encrypted blob which self-decrypts itself during upon each execution

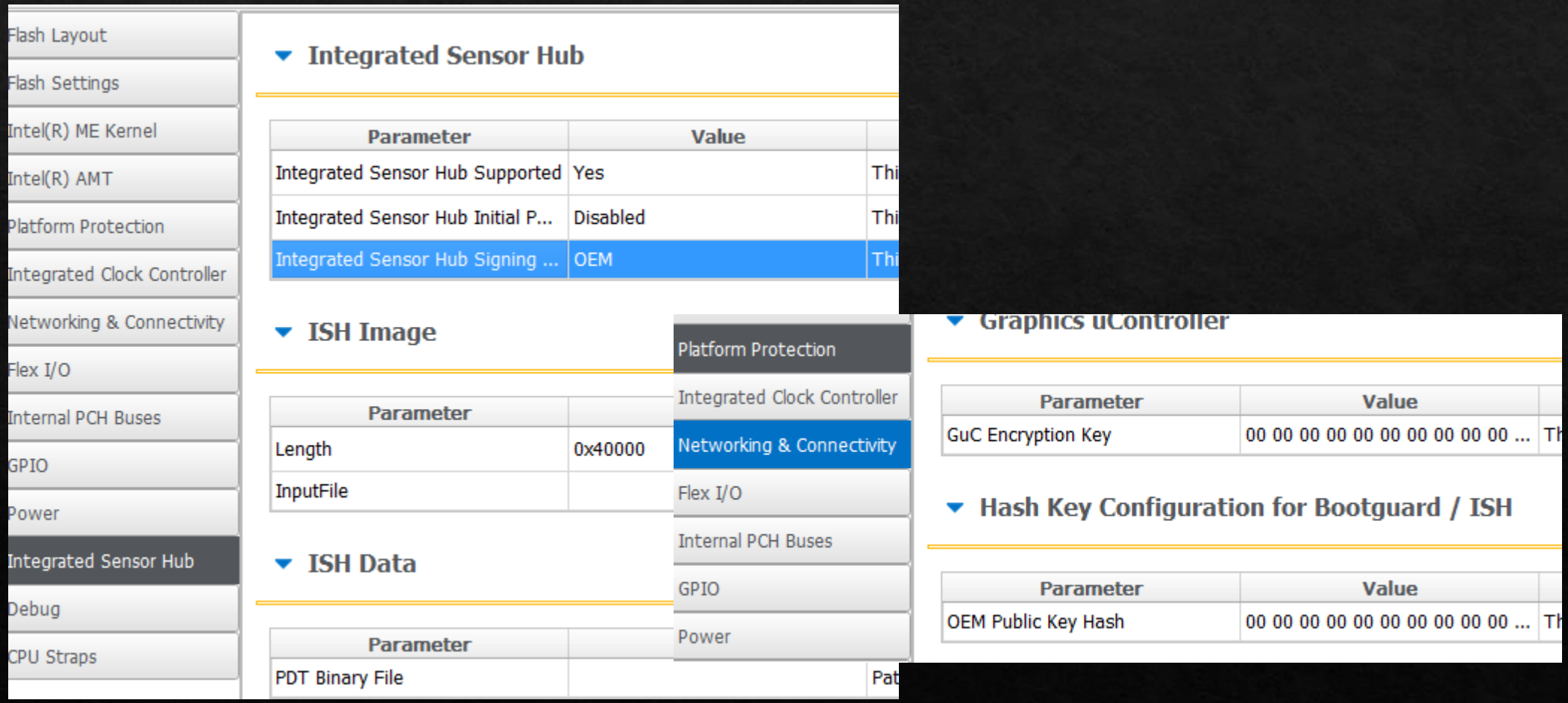
Safeguarding rootkits

Hence, the issue allows:

- to create hidden, black box and irremovable (even with SPI flash programmer) rootkit on a platform

- to modify the ISH firmware on the platform which opens a new attack surface

Safeguarding rootkits



Conclusion

Conclusion

- The description of Intel Boot Guard implementation
- A scenario to make any past BIOS modification permanent and updatable only from BG Root Key owner
- There are so many proprietary Intel blobs executing before RESET-vector
- The number of execution environments is increasing (CPU x86_64, ME x86, ISH x86, PMC ARC, …)

Mitigations

- Vendors that intentionally left the closemnf fuse unset in servicing purposes should find another way
- Vendors that left the closmnf fuse by mistake should roll out a fix (Lenovo have already done this)

- Users can disable the Intel BG technology manually:

  Just run the MEinfo to make sure the Intel BG in not configured on the platform and run the FPT with –closemnf argument

Mitigations

```
OEM Public Key Hash FPF                         Not set
OEM Public Key Hash ME
0000000000000000000000000000000000000000000000000000000000000000
ACM SVN FPF                                     0x0
KM SVN FPF                                      0x0
BSMM SVN FPF                                    0x0
GuC Encryption Key FPF                          Not set
GuC Encryption Key ME
0000000000000000000000000000000000000000000000000000000000000000

                                                FPF                     ME
                                                ---                     --
Force Boot Guard ACM                            Not set                 Disabled
Protect BIOS Environment                        Not set                 Disabled
CPU Debugging                                   Not set                 Enabled
BSP Initialization                              Not set                 Enabled
Measured Boot                                   Not set                 Disabled
Verified Boot                                   Not set                 Disabled
Key Manifest ID                                 Not set                 0x0
Enforcement Policy                              Not set                 0x0
PTT                                             Not set                 Enabled
EK Revoke State                                 Not Revoked
PTT RTC Clear Detection FPF                     Not set
```

Mitigations

```
OEM Public Key Hash FPF
0000000000000000000000000000000000000000000000000000000000000000
OEM Public Key Hash ME
0000000000000000000000000000000000000000000000000000000000000000
ACM SVN FPF                                     0x0
KM SVN FPF                                      0x0
BSMM SVN FPF                                    0x0
GuC Encryption Key FPF
0000000000000000000000000000000000000000000000000000000000000000
GuC Encryption Key ME
0000000000000000000000000000000000000000000000000000000000000000

                                                FPF                     ME
                                                ---                     --
Force Boot Guard ACM                            Disabled                Disabled
Protect BIOS Environment                        Disabled                Disabled
CPU Debugging                                   Enabled                 Enabled
BSP Initialization                              Enabled                 Enabled
Measured Boot                                   Disabled                Disabled
Verified Boot                                   Disabled                Disabled
Key Manifest ID                                 0x0                     0x0
Enforcement Policy                              0x0                     0x0
PTT                                             Enabled                 Enabled
PTT Lockout Override Counter                    0x0
EK Revoke State                                 Not Revoked
```

Thank You