ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/268389151

Platform Firmware Security

Presentation · December 2013

DOI: 10.13140/2.1.4004.8326

CITATIONS
0

READS
97

1 author:

Vincent Zimmer
Intel

384 PUBLICATIONS **305** CITATIONS

SEE PROFILE

Available from: Vincent Zimmer
Retrieved on: 06 September 2016

Platform Firmware Security

Vincent Zimmer

December 14, 2013

Usual disclaimer-
These foils and opinions are mine and not necessarily those of my employer

Who am I?

Presently a principal engineer at Intel

On the EFI/edk2 core team at Intel since 1999

Boot *firmware* at Intel starting in 1997

BIOS + SCADA/real-time *firmware* + RAID firmware since 1992 (ah, those days in TX....)

Some chores include SMM, low level SI init (PEI), EFI TPM measured boot (SRTM), evolution of network boot (PXE into netboot6), ***UEFI Secure Boot***

Catch me at vincent.zimmer@intel.com, vincent.zimmer@gmail.com, sites.google.com/site/vincentzimmer, Twitter @vincentzimmer

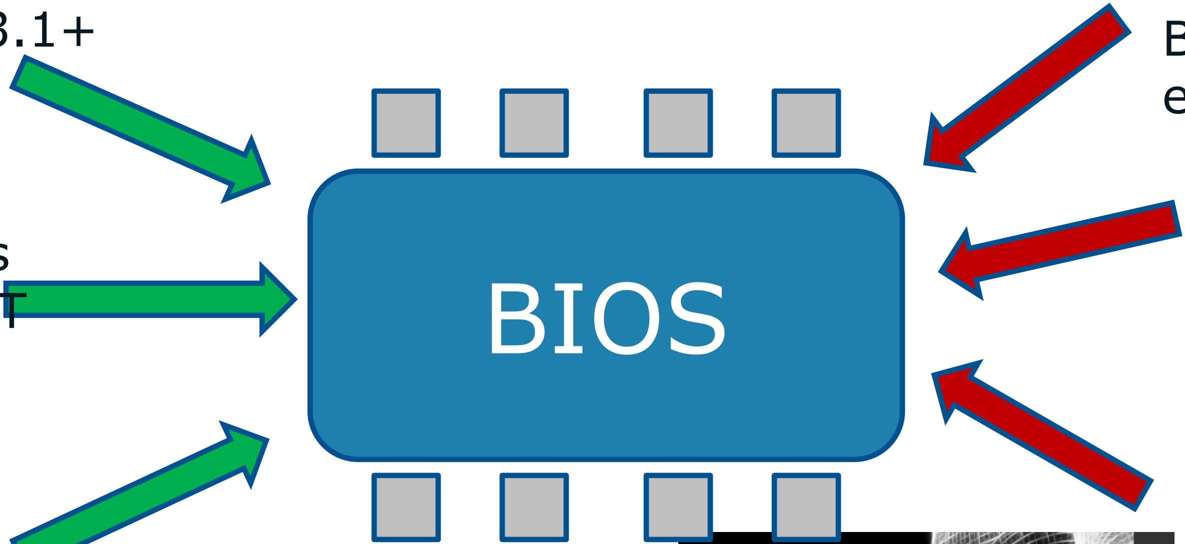
Pressure on BIOS

Industry requirements (ex. UEFI 2.3.1+ Ch 27, TCG)

Government requirements (ex: US NIST SP800-147)

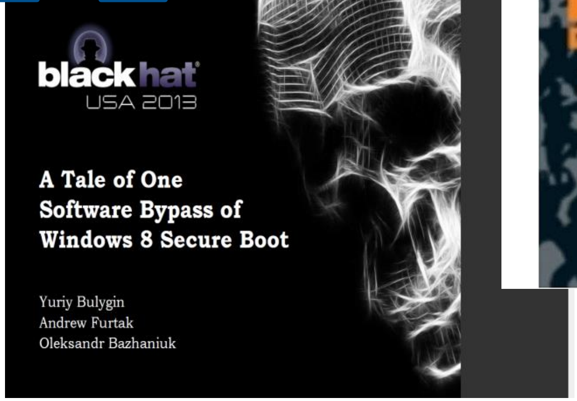
Product dvlp requirements (ex. SDL)

Customers requiring security (ex. US DoD, Corporate IT)



Malware (ex. Chernobyl, 2000 Bootkits, 2011 etc)

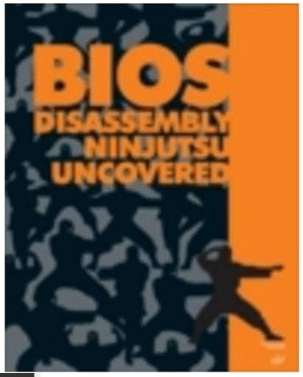
Researchers (ex. Invisible Things Lab BMP attacks 2004)



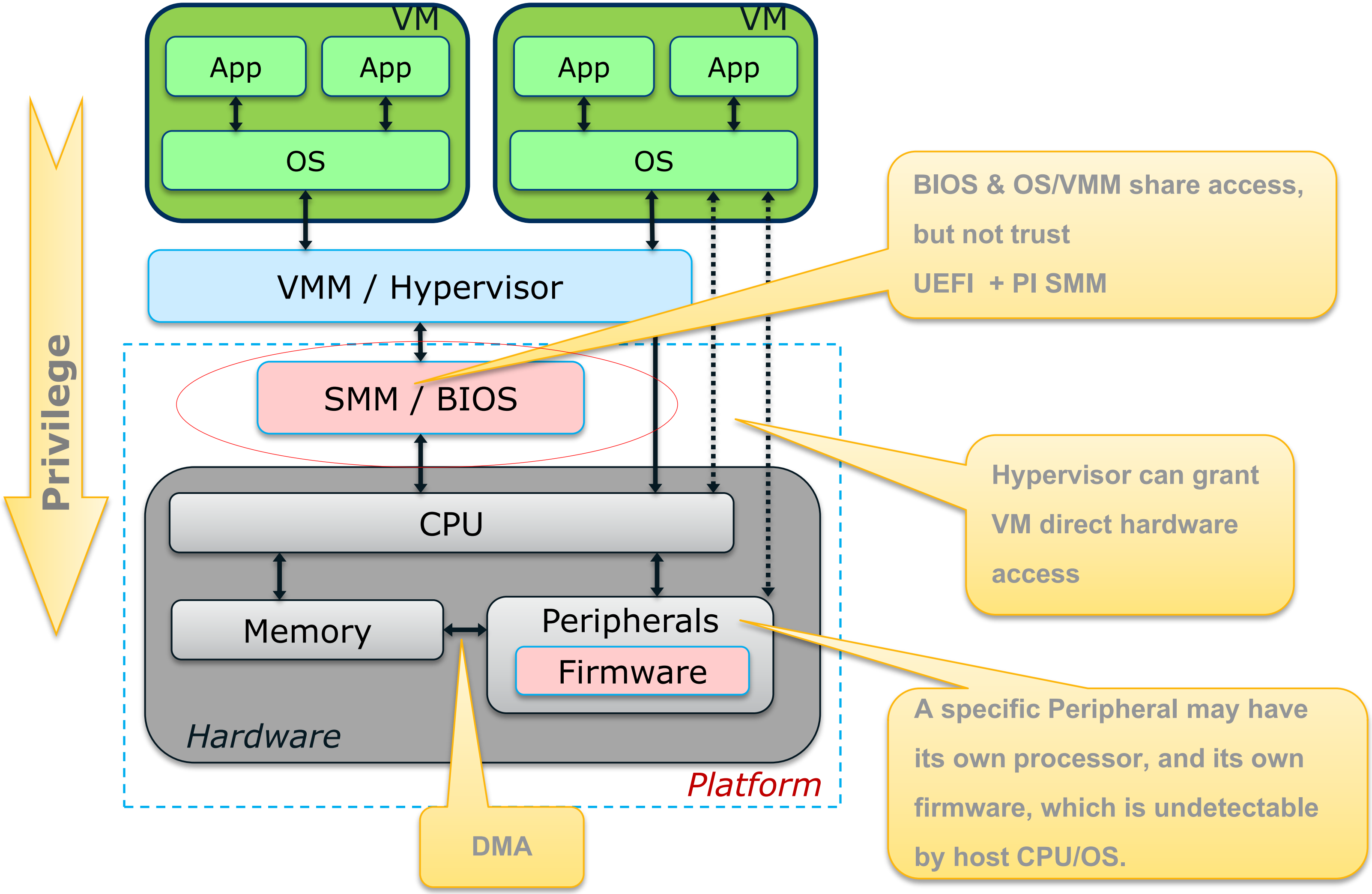
black hat USA 2013

A Tale of One Software Bypass of Windows 8 Secure Boot

Yuriy Bulygin
Andrew Furtak
Oleksandr Bazhaniuk



Where are we (BIOS / UEFI firmware)?



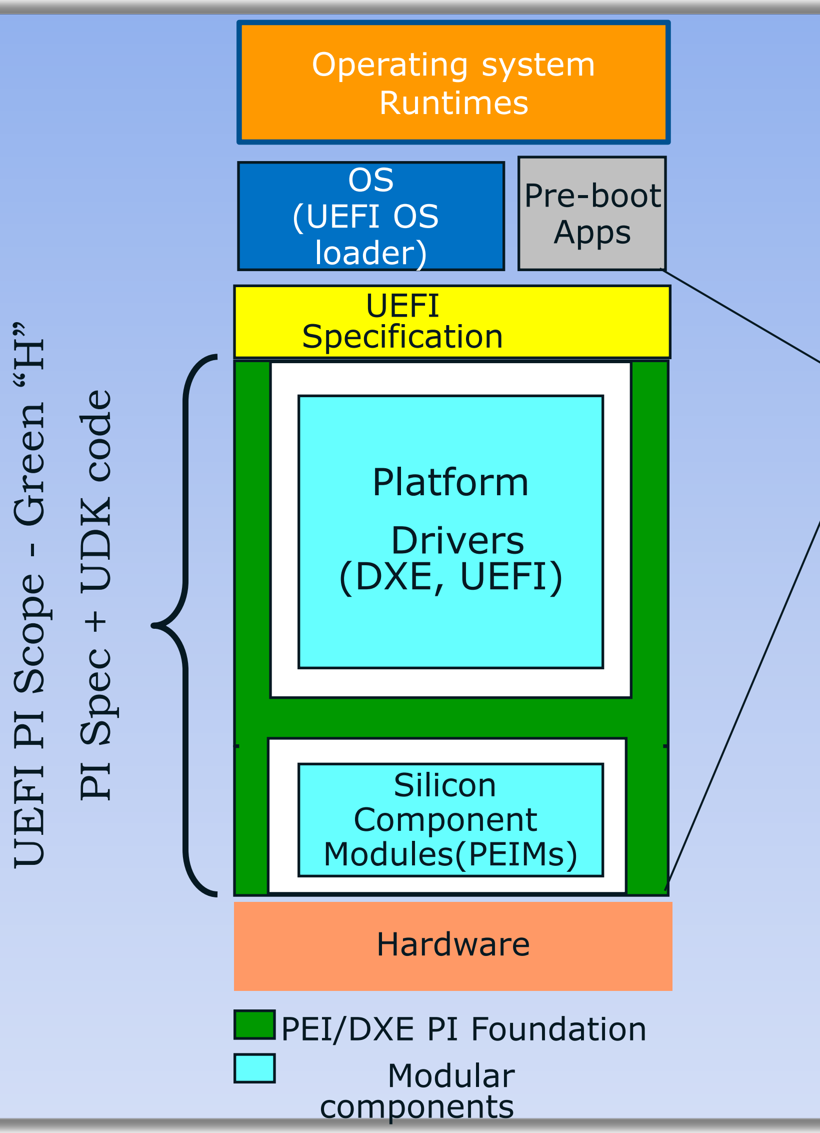
BIOS & OS/VMM share access, but not trust

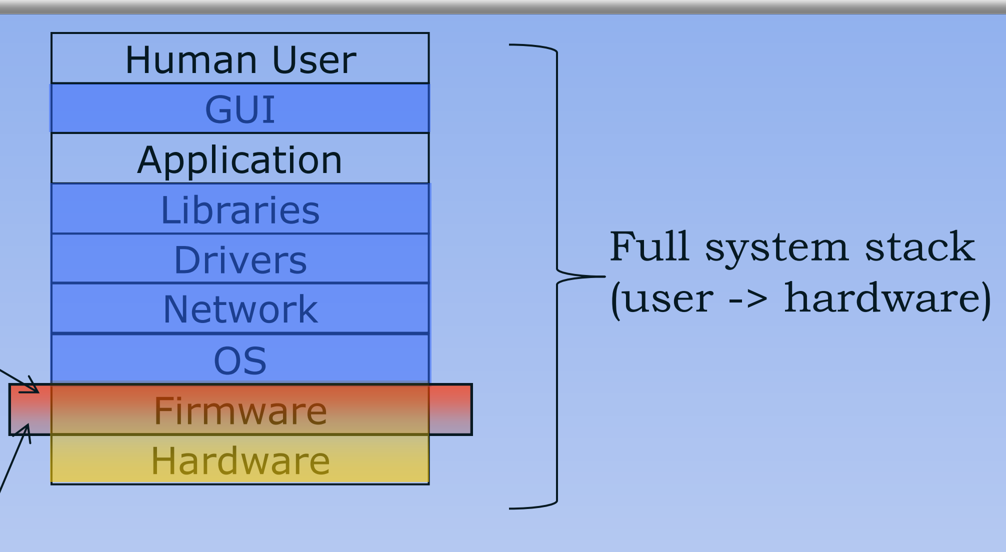
UEFI + PI SMM

Hypervisor can grant VM direct hardware access

A specific Peripheral may have its own processor, and its own firmware, which is undetectable by host CPU/OS.

DMA





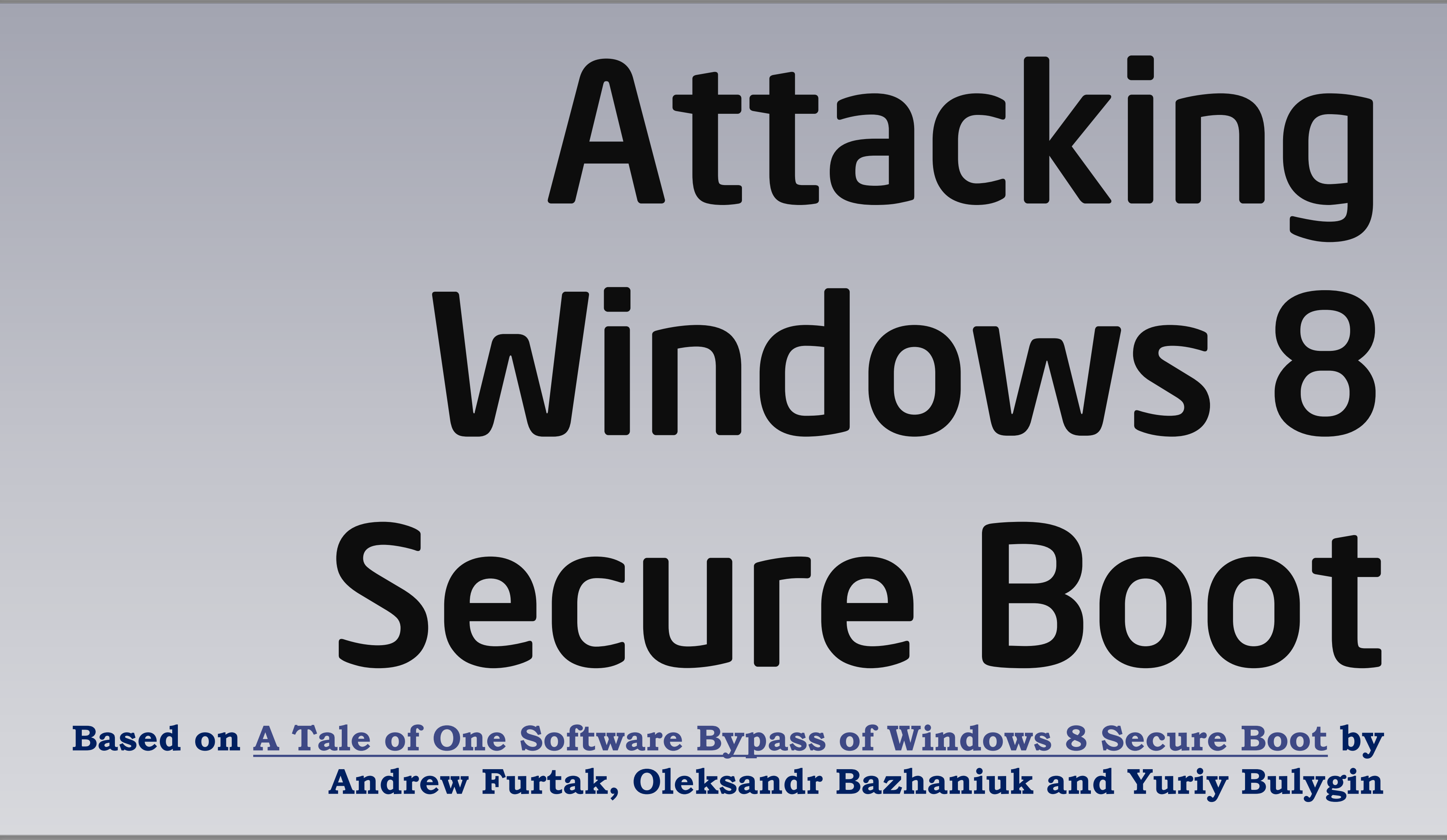
- UEFI 2.4 specifies how firmware boots OS loader and drivers– **3rd party extensible**
- UEFI's Platform Initialization (PI) 1.2.1
- Architecture specifies how Driver Execution Environment (DXE) Drivers and Pre-EFI Initialization (PEI) Modules (PEIMs) initialize Silicon and the platform
- DXE is preferred UEFI Implementation
- PEIMs, UEFI and DXE drivers implements networking, Update, other security features - **Only Update by the OEM**

Building UEFI – Platform Initialization (PI)

UEFI / PI is a type of BIOS
BIOS– aka. the Rodney Dangerfield of Software

“No respect”

Offense/Attack (not today's talk)

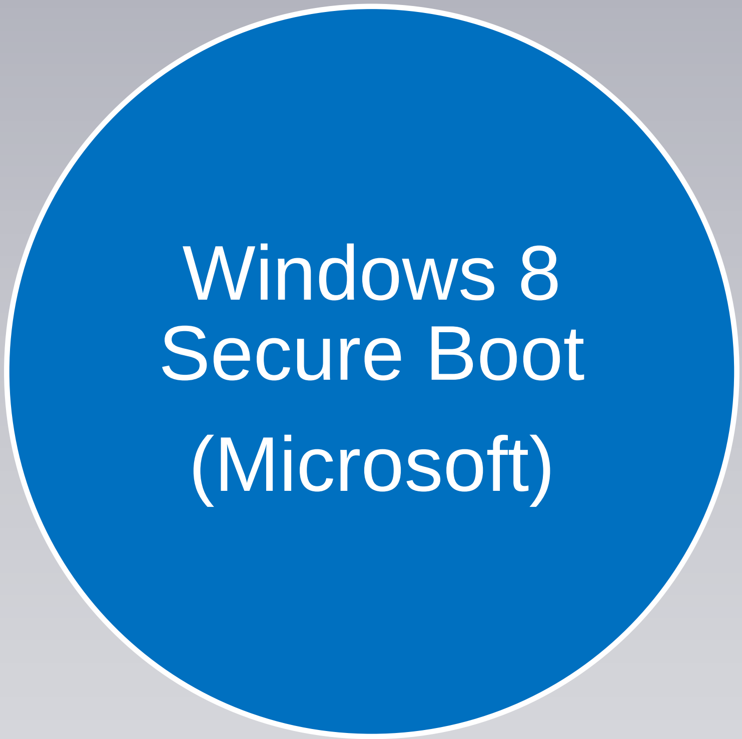


Attacking Windows 8 Secure Boot

Based on A Tale of One Software Bypass of Windows 8 Secure Boot by Andrew Furtak, Oleksandr Bazhaniuk and Yuriy Bulygin

http://www.c7zero.info/stuff/Windows8SecureBoot_Bulygin-Furtak-Bazhniuk_BHUSA2013.pdf

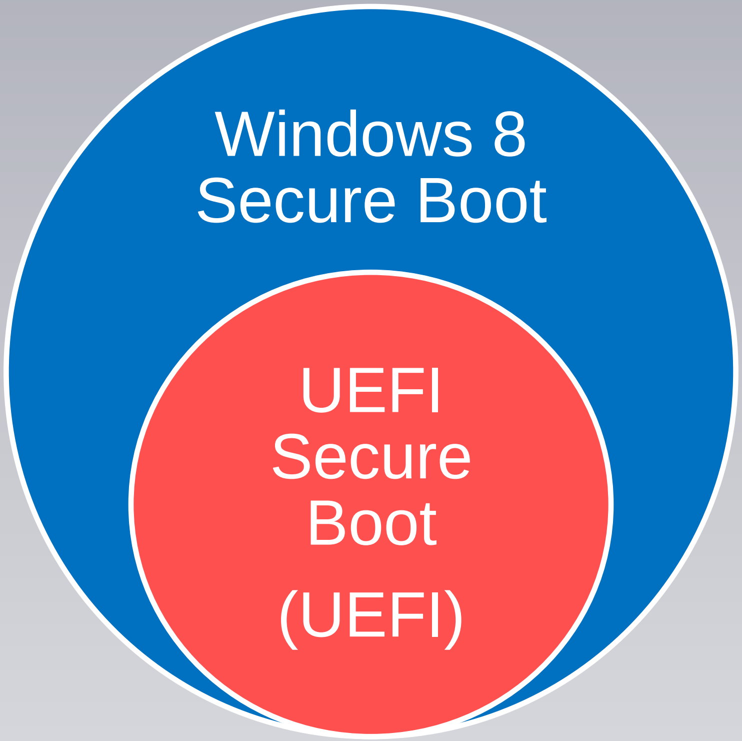
We think Windows 8 Secure Boot looks like this



Windows 8
Secure Boot

(Microsoft)

Or more like this



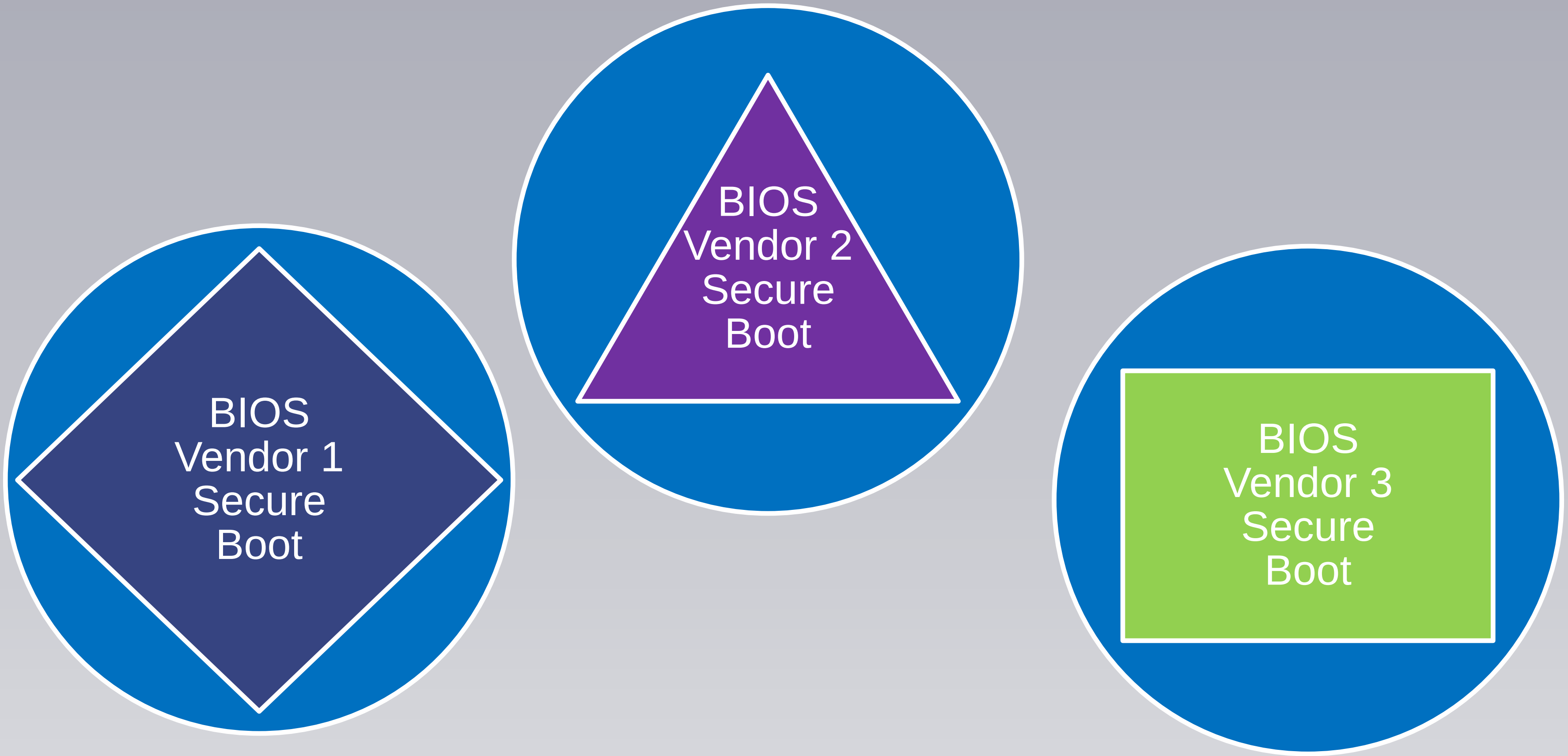
Windows 8
Secure Boot

UEFI
Secure
Boot

(UEFI)

Ideal

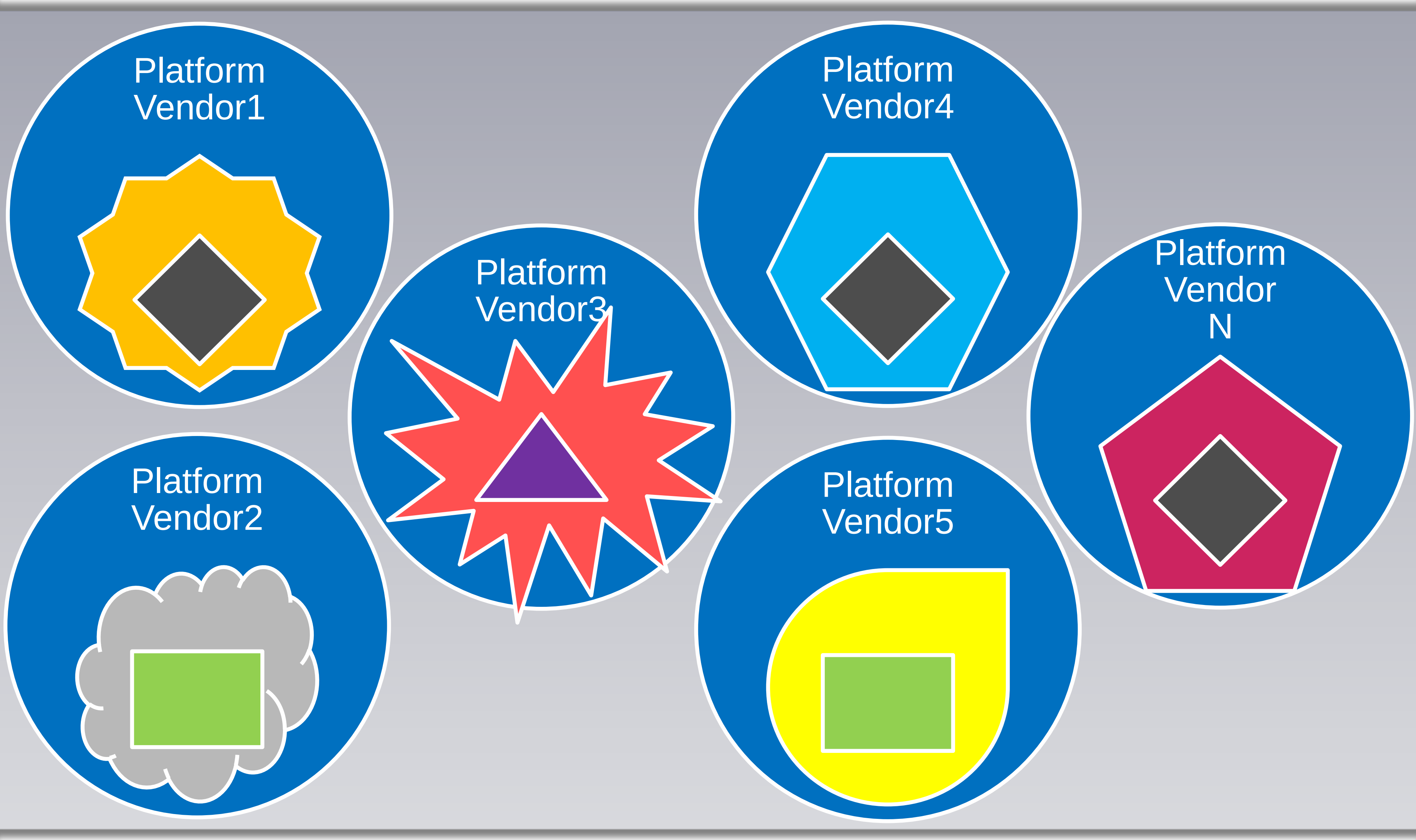
How exciting! … But still not close



BIOS Vendor 2 Secure Boot

BIOS Vendor 1 Secure Boot

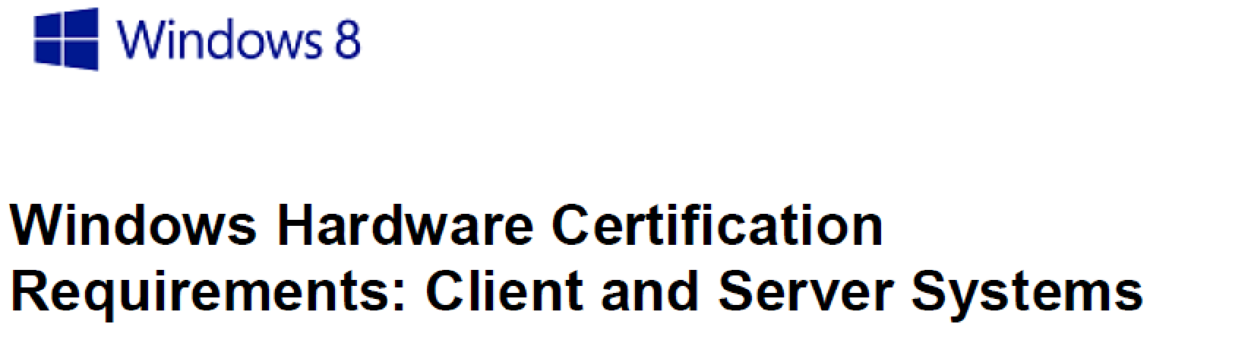
BIOS Vendor 3 Secure Boot



The Reality Is Much More Exciting

Windows 8 Secure Boot is only secure when ALL platform/BIOS vendors do a couple of things correctly

- Allow signed UEFI firmware updates only
- Protect UEFI firmware in SPI flash from direct modification
- Protect firmware update components (inside SMM or DXE on reboot)
- Program SPI controller and flash descriptor securely
- Protect SecureBootEnable/CustomMode/PK/KEK/db(x) in NVRAM
- Implement VariableAuthenticated in SMM and physical presence checks
- Protect SetVariable runtime API
- Securely disable Compatibility Support Module (CSM), unsigned legacy Option ROMs and MBR boot loaders
- Configure secure image verification policies (no ALLOW_EXECUTE)
- Build platform firmware using latest UEFI/EDK sources
- Correctly implement signature verification and crypto functionality
- **And don’t introduce a single bug in all of this…**



Windows 8

Windows Hardware Certification Requirements: Client and Server Systems

System.Fundamentals.Firmware.UEFISecureBoot

3 When Secure Boot is Enabled, CSM must NOT be loaded

7 Secure Boot must be rooted in a protected or ROM-based Public Key

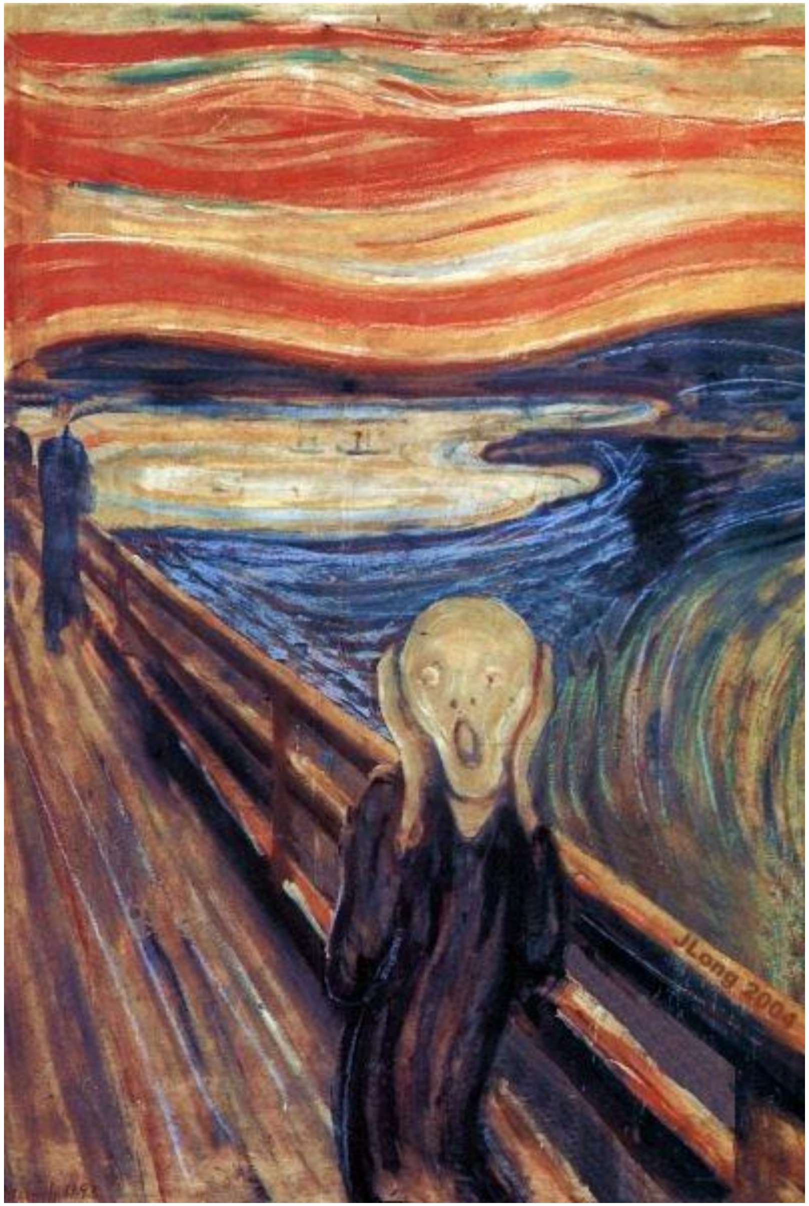
8 Secure firmware update process

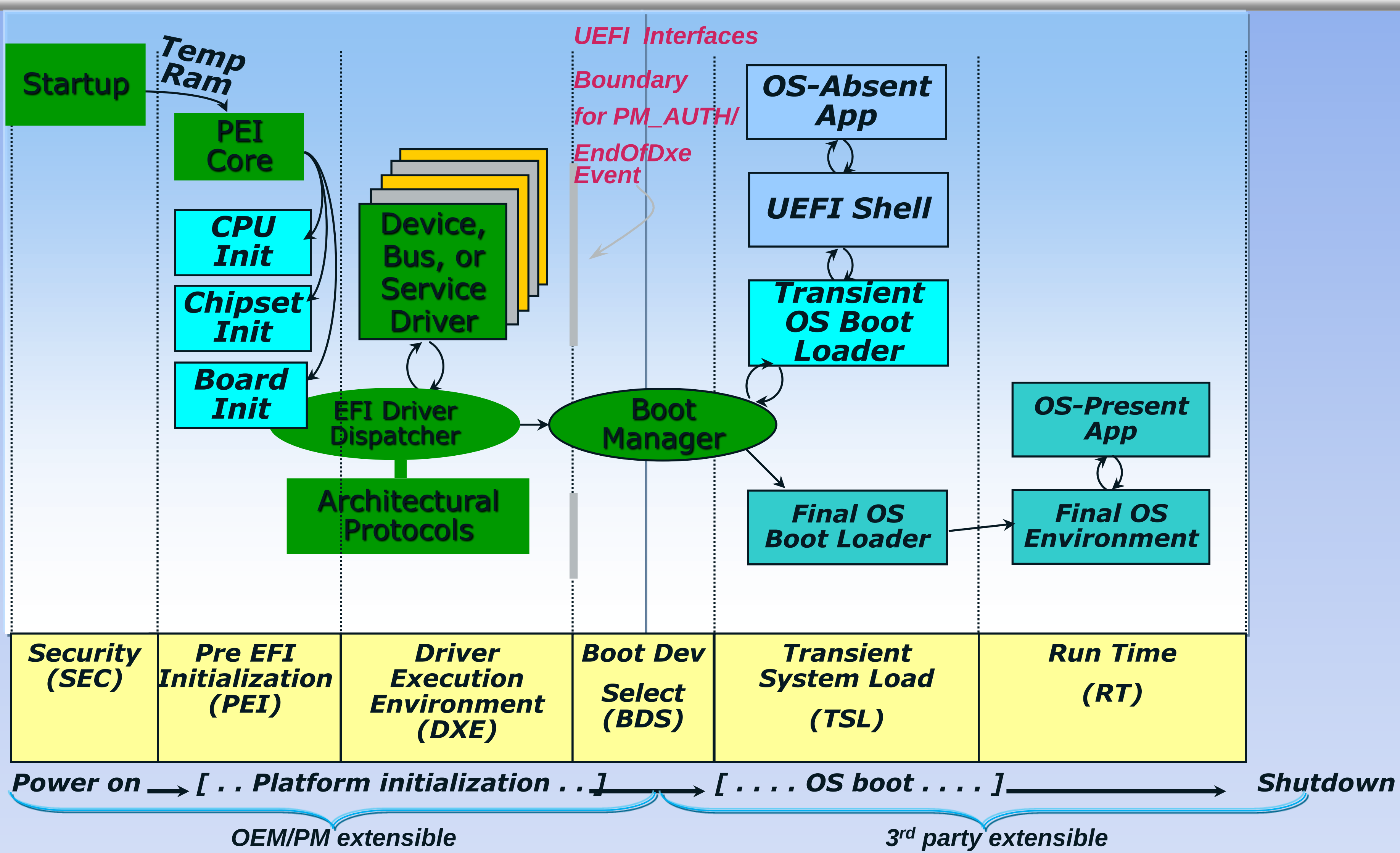
9 Signed Firmware Code Integrity Check

14 No in-line mechanism is provided whereby a user can bypass Secure Boot failures and boot anyway

…

Windows 8 Secure Boot Requirements


JLong 2004



UEFI Interfaces

Startup

Temp Ram

PEI Core

CPU Init

Chipset Init

Board Init

Device, Bus, or Service Driver

EFI Driver Dispatcher

Architectural Protocols

Boundary for PM_AUTH/ EndOfDxe Event

Boot Manager

OS-Absent App

UEFI Shell

Transient OS Boot Loader

Final OS Boot Loader

OS-Present App

Final OS Environment

Security (SEC)	Pre EFI Initialization (PEI)	Driver Execution Environment (DXE)	Boot Dev Select (BDS)	Transient System Load (TSL)	Run Time (RT)

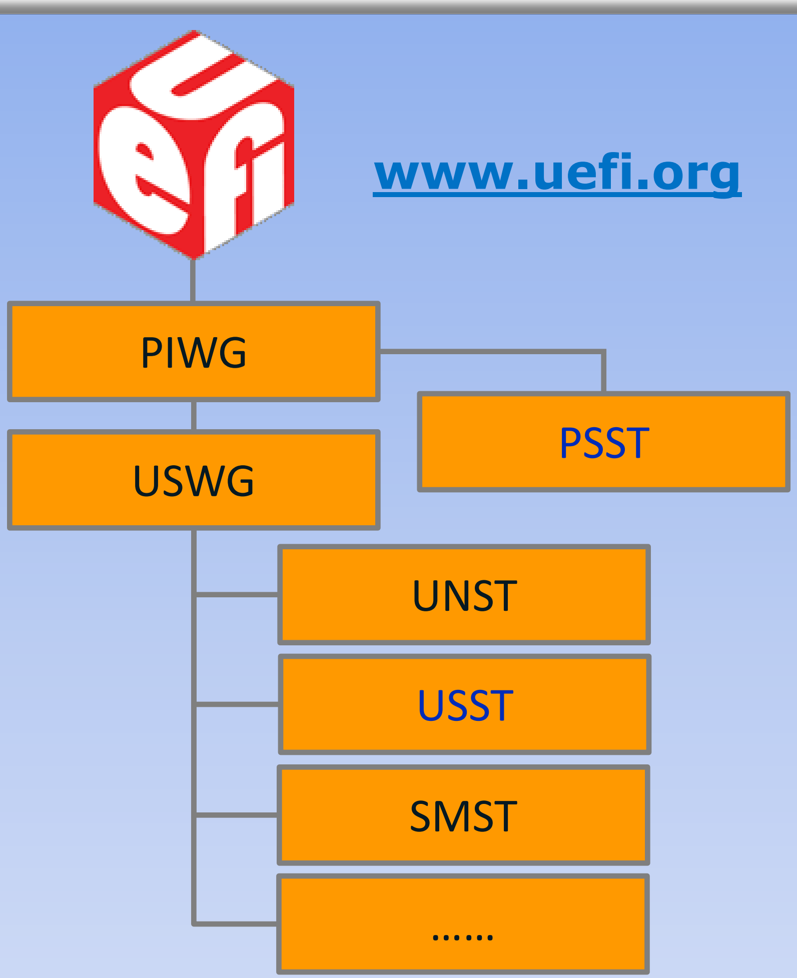
Power on → [. . Platform initialization . .] → [. . . . OS boot . . . .] → Shutdown

OEM/PM extensible

3rd party extensible

Overall Boot Timeline

Security Working Groups in UEFI



www.uefi.org

PIWG

PSST

USWG

UNST

USST

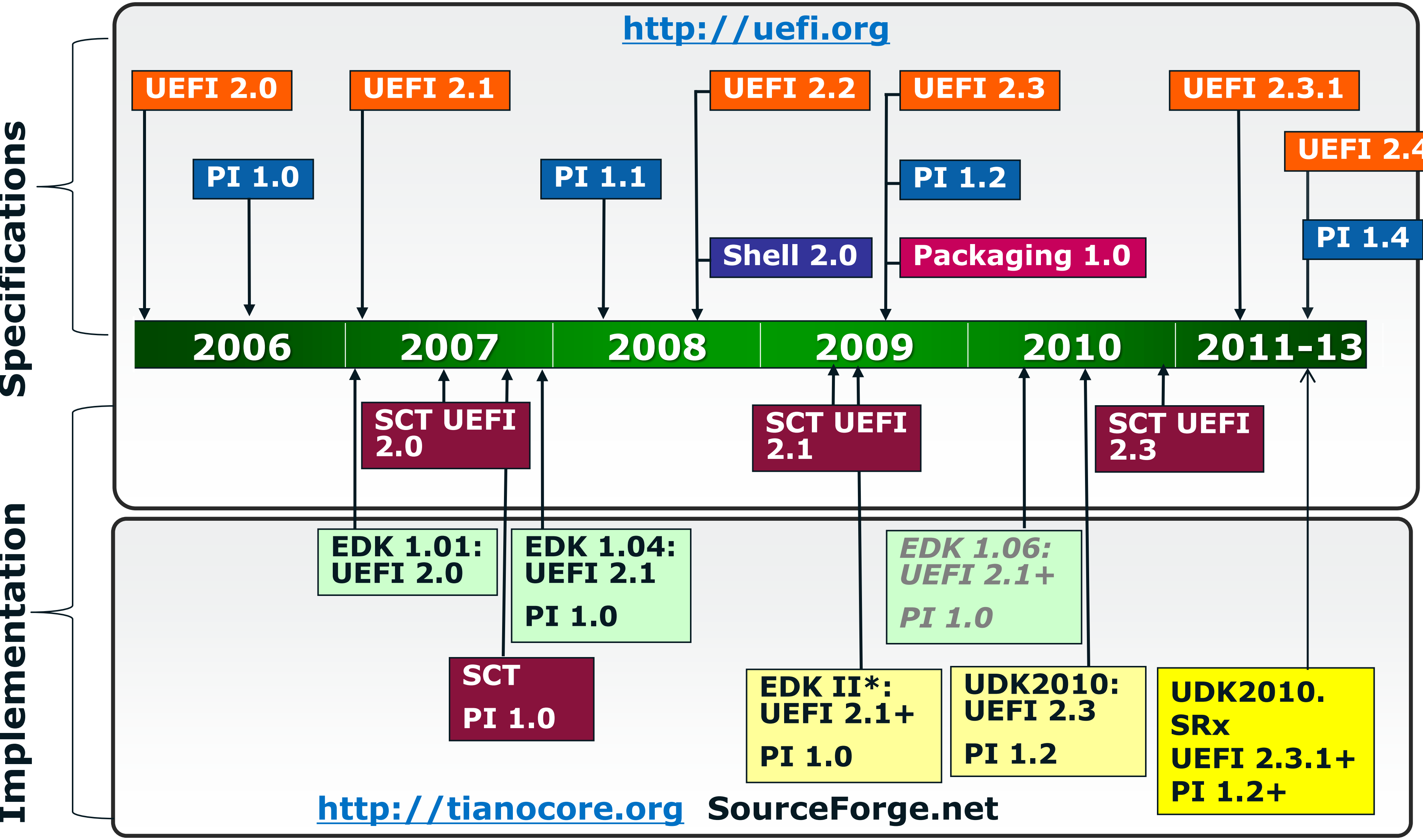
SMST

……

Note: Engaged in firmware/boot
Related WG’s of Trusted Computing Group (TCG), IETF, DMTF

- **USST**
  - **U**SWG **S**ecurity **S**ub-**t**eam
  - Chaired by Vincent Zimmer (Intel)
  - Responsible for all security related material and the team has been responsible for the added security infrastructure in the UEFI
- **PSST**
  - PIWG Security Sub-team
  - Chaired by Vincent Zimmer (Intel)
  - Produce design guide(s) that define integrity protection business goals, provide a security model within which these goals are expressed as security requirements, and identify architectural and implementation issues that cause the requirements not to be met.
- **UNST**
  - UEFI Network Sub-team
  - Chaired by Vincent Zimmer (Intel)
  - Evolve network boot & network security infrastructure for UEFI Specification

Specification & Tianocore.org Timeline



http://uefi.org

Specifications

UEFI 2.0

PI 1.0

UEFI 2.1

PI 1.1

UEFI 2.2

Shell 2.0

UEFI 2.3

PI 1.2

Packaging 1.0

UEFI 2.3.1

UEFI 2.4

PI 1.4

2006

2007

2008

2009

2010

2011-13

SCT UEFI 2.0

SCT UEFI 2.1

SCT UEFI 2.3

Implementation

EDK 1.01:
UEFI 2.0

EDK 1.04:
UEFI 2.1
PI 1.0

EDK 1.06:
UEFI 2.1+
PI 1.0

SCT
PI 1.0

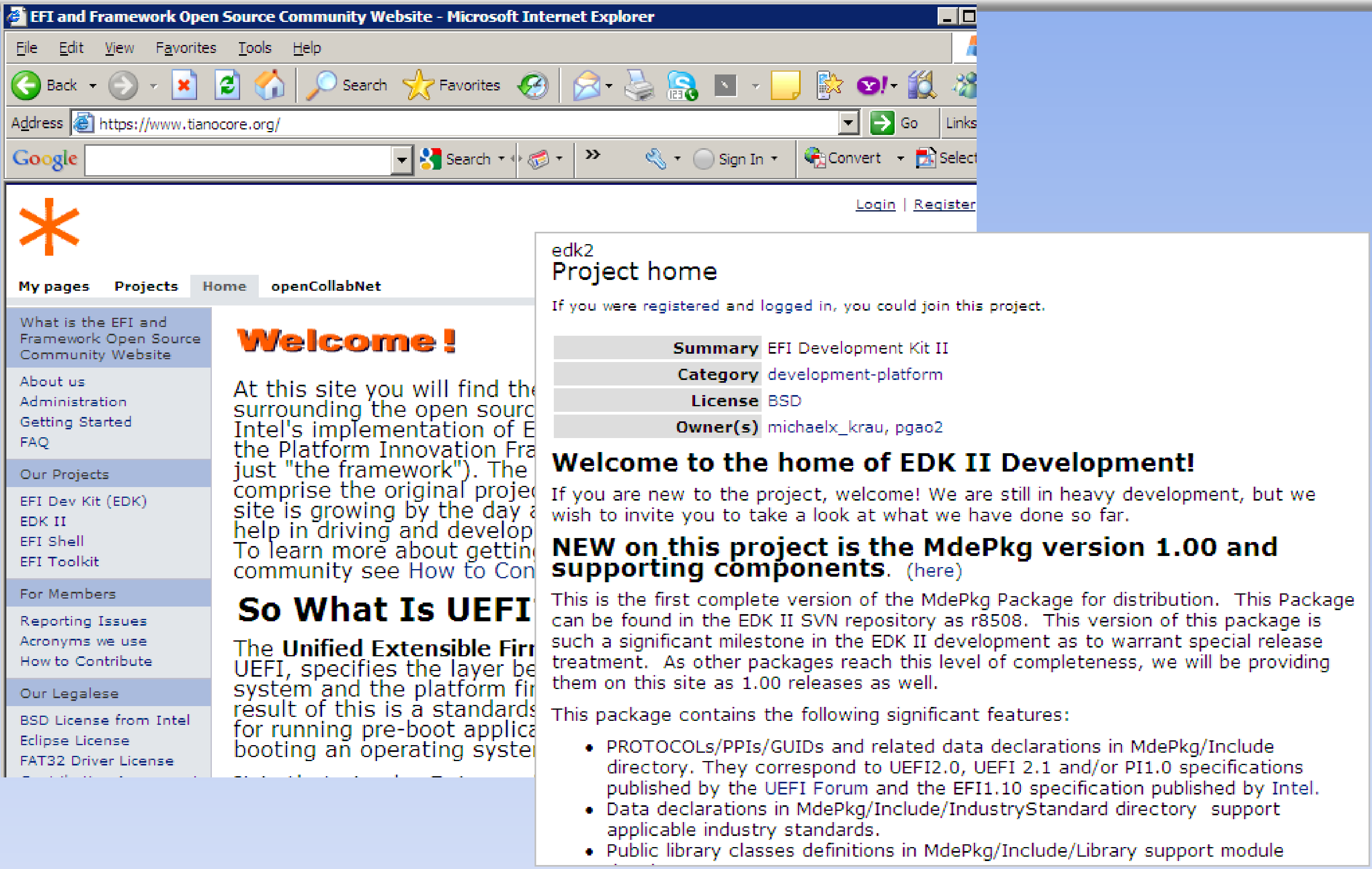
EDK II*:
UEFI 2.1+
PI 1.0

UDK2010:
UEFI 2.3
PI 1.2

UDK2010.
SRx
UEFI 2.3.1+
PI 1.2+

http://tianocore.org SourceForge.net

All products, dates, and programs are based on current expectations and subject to change without notice.


EFI and Framework Open Source Community Website - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Address https://www.tianocore.org/
Go
Links
Google
Search
Sign In
Convert
Select
Login | Register
My pages
Projects
Home
openCollabNet

What is the EFI and Framework Open Source Community Website

About us
Administration
Getting Started
FAQ

Our Projects

EFI Dev Kit (EDK)
EDK II
EFI Shell
EFI Toolkit

For Members

Reporting Issues
Acronyms we use
How to Contribute

Our Legalese

BSD License from Intel
Eclipse License
FAT32 Driver License

Welcome!

At this site you will find the surrounding the open sourc Intel's implementation of E the Platform Innovation Fra just "the framework"). The comprise the original projec site is growing by the day a help in driving and develop To learn more about gettin community see How to Con

So What Is UEFI

The **Unified Extensible Firm** UEFI, specifies the layer be system and the platform fir result of this is a standards for running pre-boot applica booting an operating syste

edk2

Project home

If you were registered and logged in, you could join this project.

Summary	EFI Development Kit II
Category	development-platform
License	BSD
Owner(s)	michaelx_krau, pgao2

Welcome to the home of EDK II Development!

If you are new to the project, welcome! We are still in heavy development, but we wish to invite you to take a look at what we have done so far.

NEW on this project is the MdePkg version 1.00 and supporting components. (here)

This is the first complete version of the MdePkg Package for distribution. This Package can be found in the EDK II SVN repository as r8508. This version of this package is such a significant milestone in the EDK II development as to warrant special release treatment. As other packages reach this level of completeness, we will be providing them on this site as 1.00 releases as well.

This package contains the following significant features:

- PROTOCOLs/PPIs/GUIDs and related data declarations in MdePkg/Include directory. They correspond to UEFI2.0, UEFI 2.1 and/or PI1.0 specifications published by the UEFI Forum and the EFI1.10 specification published by Intel.
- Data declarations in MdePkg/Include/IndustryStandard directory support applicable industry standards.
- Public library classes definitions in MdePkg/Include/Library support module

UDK2010 Available on Tianocore.org

How to build it? UDK2010

Industry Standards Compliance

- UEFI 2.0, UEFI 2.1, UEFI 2.2, UEFI 2.3; PI 1.0, PI 1.1, PI 1.2

Extensible Foundation for Advanced Capabilities

- Pre-OS Security
- Rich Networking
- Manageability

Support for UEFI Packages

- Import/export modules source/binaries to many build systems

Maximize Re-use of Source Code**

- Platform Configuration Database (PCD) provides "knobs" for binaries
- ECP provides for reuse of EDK1117 (EDK I) modules
- Improved modularity, library classes and instances
- Optimize for size or speed

Multiple Development Environments and Tool Chains**

- Windows, Linux, OSX
- VS2003, VS2005, WinDDK, Intel, GCC

Fast and Flexible Build Infrastructure**

- 4X+ Build Performance Improvement (vs EDKI)
- Targeted Module Build Flexibility

*** benefit of EDK II codebase*

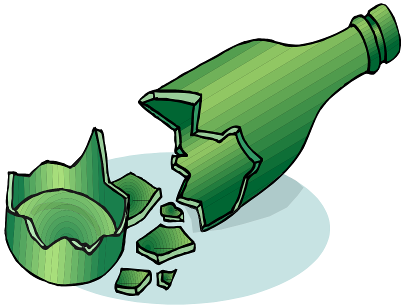
Maximize the open source at www.tianocore.org

Why use UEFI Secure Boot

Without

Possible corrupted or destroyed data

- BootKit virus – MBR Rootkits
- Network boot attacks e.g. PXESPOILT
- Code Injection Attacks



With

Data integrity

- Trusted boot to OS
- Trusted drivers
- Trusted Applications

a

*

What is Security from BIOS Perspective

Secure Boot - UEFI

- Defined a policy for Image loading
- Cryptographically signed
  - Private key at signing server
  - Public key in platform

Measured Boot -Trusted Computing Group (TCG)

- Trusted Platform Module (TPM)
  - Isolated storage and execution for Logging changes, attestation

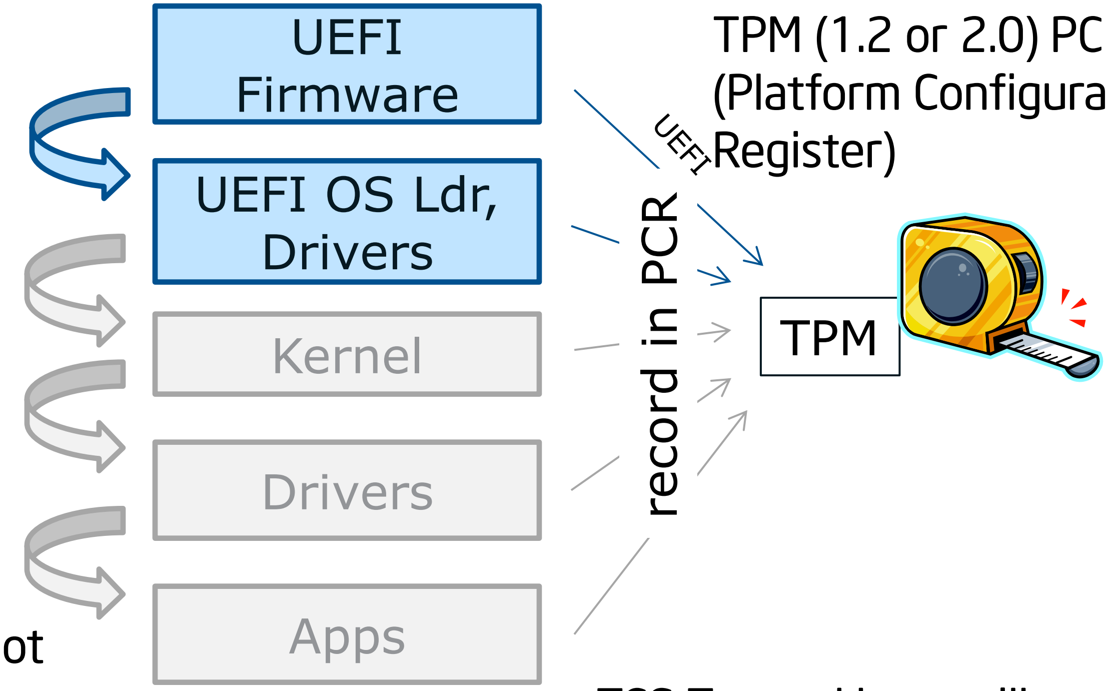
NIST 800-147 -Security Guidelines for System BIOS Implementations

UEFI Secure Boot VS TCG Trusted Boot

UEFI authenticate OS
loader
(pub key and policy)

Check signature of before loading

UEFI PI will measure OS loader & UEFI drivers into TPM (1.2 or 2.0) PCR (Platform Configuration Register)



- UEFI Secure boot will stop platform boot if signature not valid (OEM to provide remediation capability)
- UEFI will require remediation mechanisms if boot fails

- TCG Trusted boot will never fail
- Incumbent upon other SW to make security decision using attestation

NIST Implementation Requirements

Make sure UEFI PI code is protected – NIST 800-147

The NIST BIOS Protection Guidelines break down to three basic requirements…

1. The BIOS must be protected
2. BIOS updates must be signed
3. BIOS protection cannot be bypassed

UEFI Secure Boot Goals

Local verification. Complements measured boot

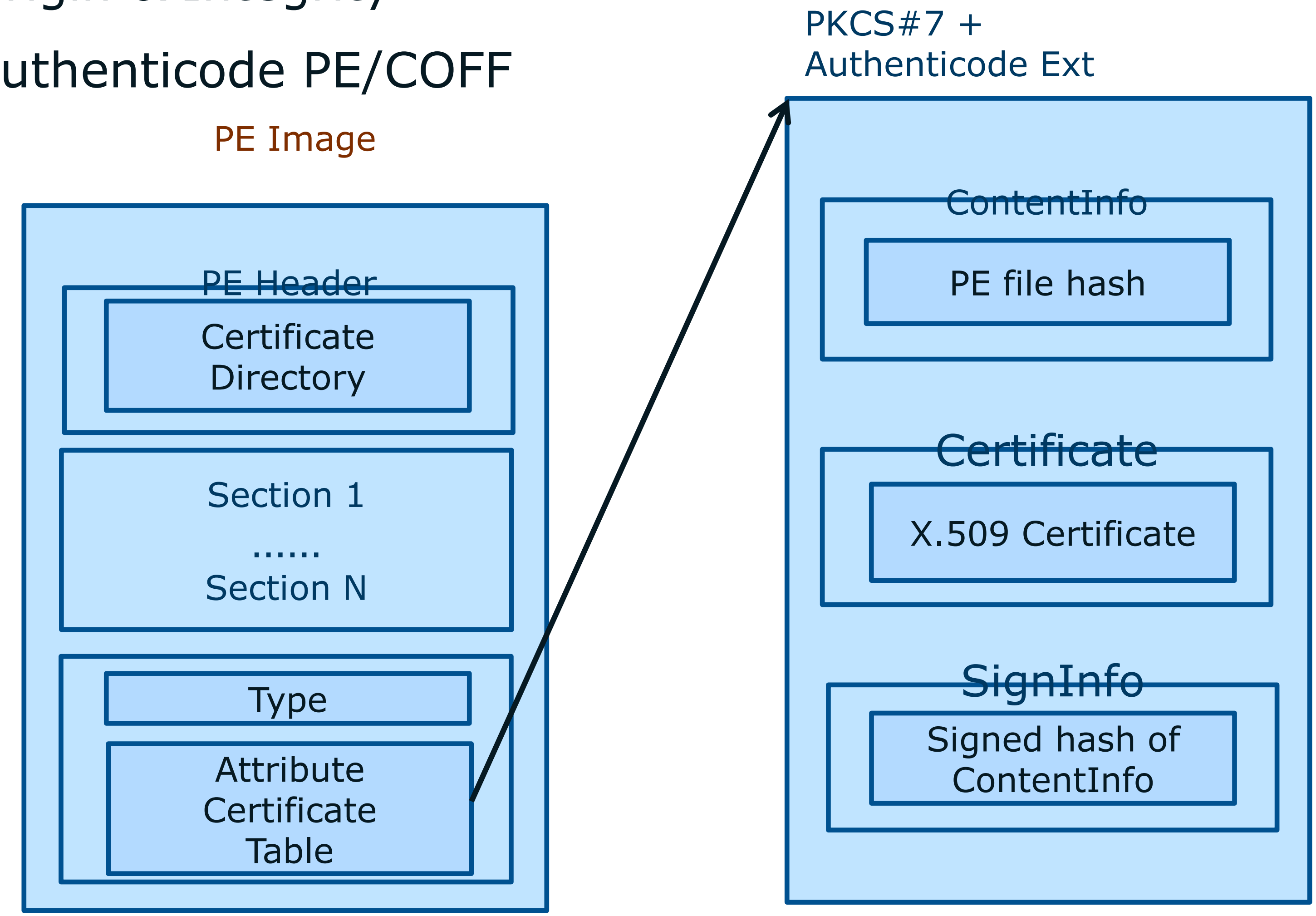
Allow the platform owner to check the integrity and security of a given UEFI image ensuring that the image is only loaded in an approved manner.

Allow the platform owner to manage the platform's security policy as defined by the UEFI Secure Boot authenticated variables

UEFI Image (driver & application/OS loader) Signing

Why? – Origin & Integrity

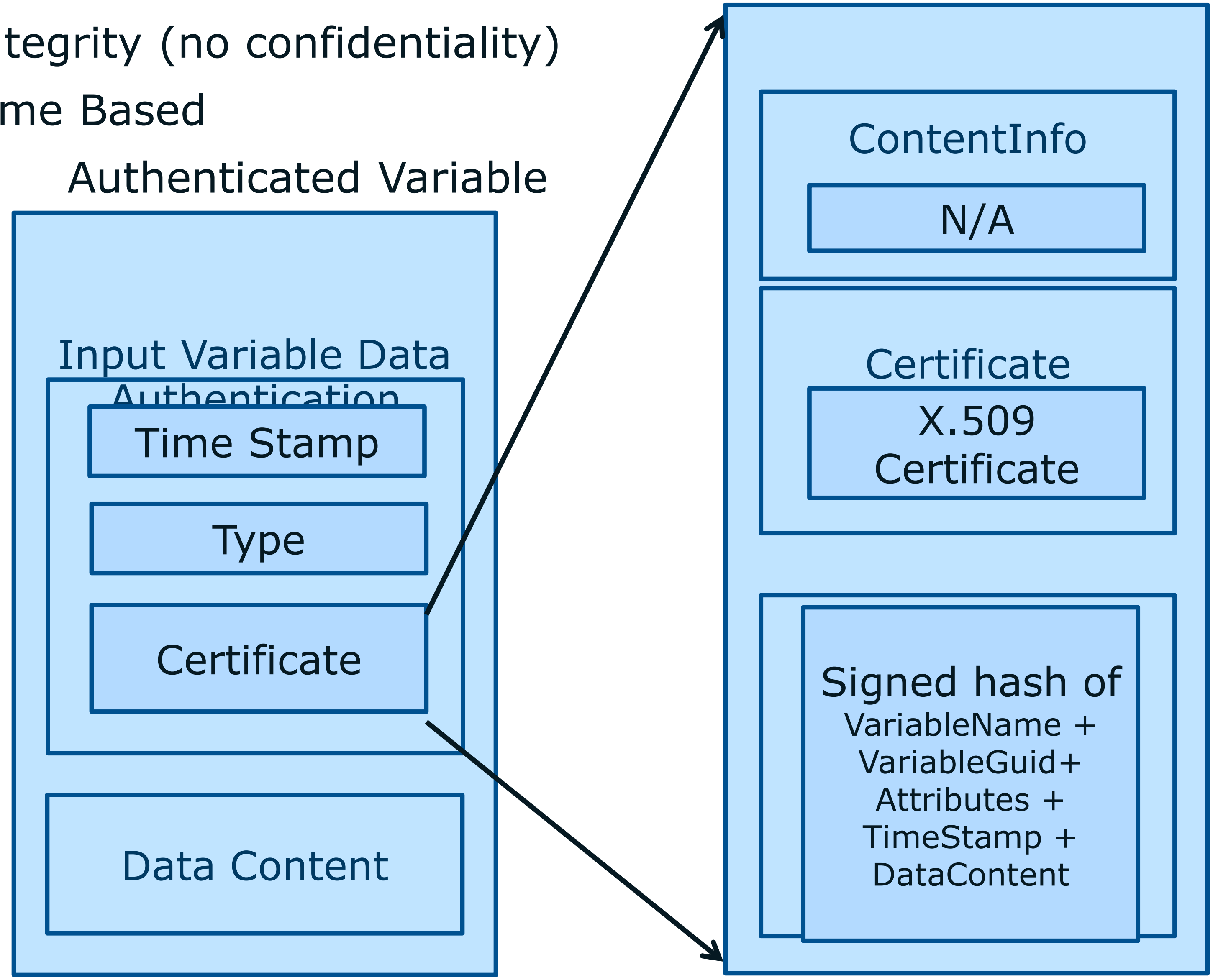
How? – Authenticode PE/COFF



Policy - UEFI Authenticated Variable

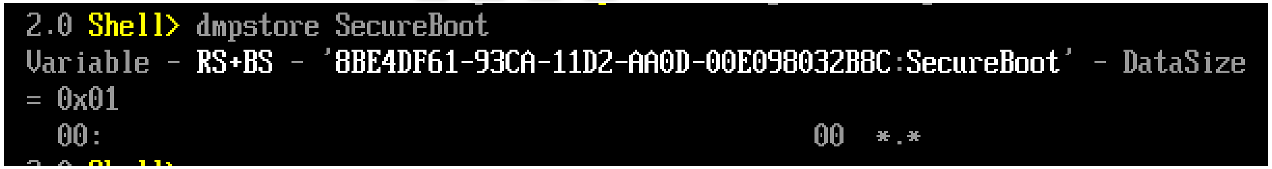
Why? – Integrity (no confidentiality)

How? – Time Based



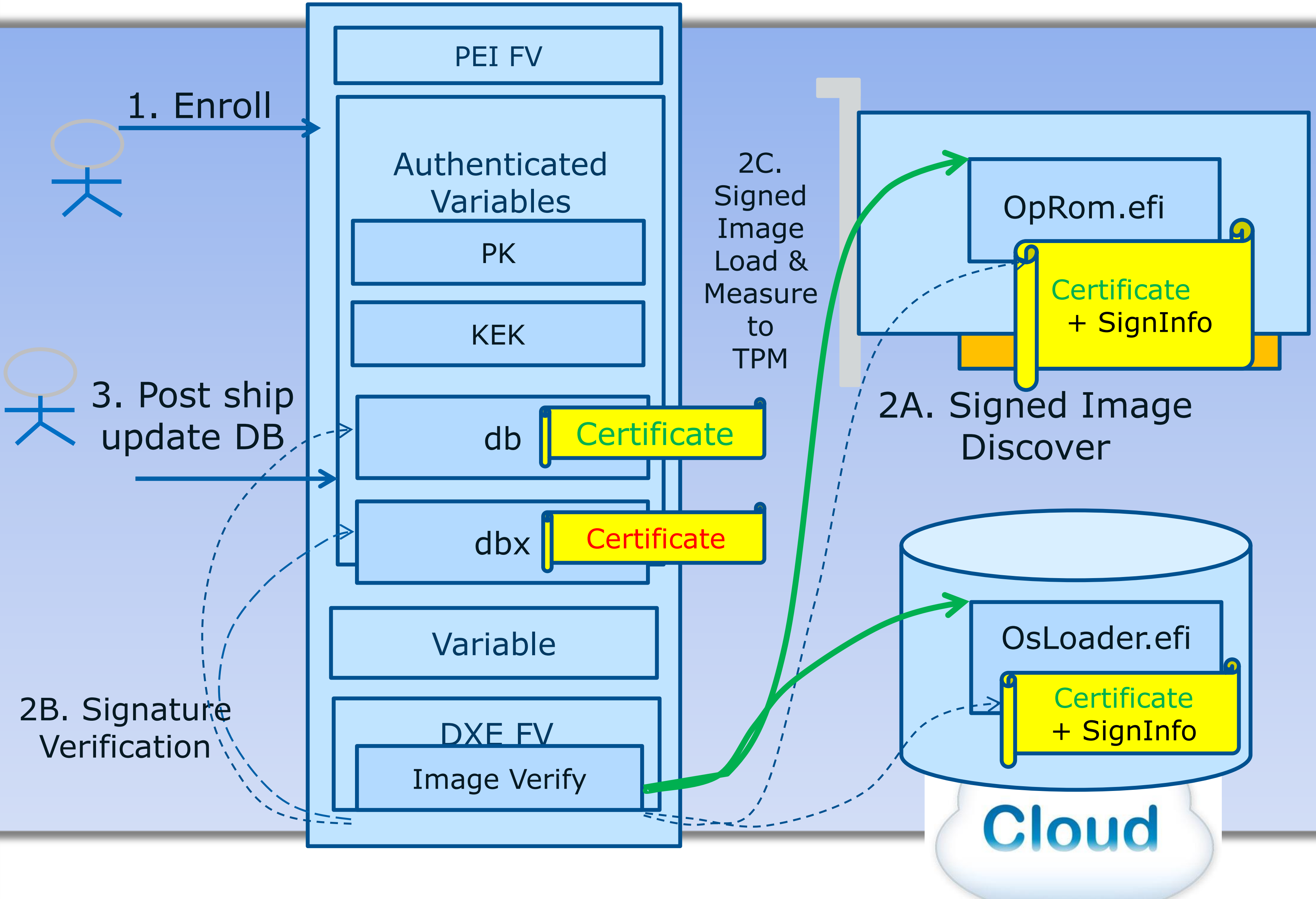
Secure Boot's Authenticated Variables

Key/ DB Name	Variable	Details
PkPub	PK	OEM and Platform FW- format is RSA-2048
Key Exchange Key	KEK	Platform FW and OS - format is RSA-2048
Authorized Signature DB	DB	Authorized Signing certificates - white list
Forbidden Signature DB	DBX	Unuthorized Signing certificates - Black list
Setup Mode	SetupMode	NULL - Secure Boot not supported 0 - PK is enrolled - in user mode User mode requires authentication 1 – Platform is in Setup mode – no PK enrolled
Secure Boot	SecureBoot	1-Platform in Secure boot mode



```
2.0 Shell> dmpstore SecureBoot
Variable - RS+BS - '8BE4DF61-93CA-11D2-AA0D-00E098032B8C:SecureBoot' - DataSize
= 0x01
  00:                                              00  *.*
2.0 Shell>
```

UEFI Secure Boot Flow



1. Enroll

PEI FV

Authenticated Variables

PK

KEK

db

Certificate

dbx

Certificate

Variable

DXE FV

Image Verify

3. Post ship update DB

2B. Signature Verification

2C. Signed Image Load & Measure to TPM

OpRom.efi

Certificate + SignInfo

2A. Signed Image Discover

OsLoader.efi

Certificate + SignInfo

Cloud

Relevant open source software packages/routines for Authorization flow

MdeModulePkg

LoadImage Boot Service

gBS->LoadImage
CoreLoadImage()

EFI_SECURITY_ARCH_PROTOCOL
SecurityStubDxe

SecurityStubAuthenticateState()

DxeSecurityManagementLib

RegisterSecurityHandler()
ExecuteSecurityHandlers()

SecurityPkg
DxeImageVerificationLib

DxeImageVerificationHandler()
HashPeImage()
HashPeImageByType()
VerifyWinCertificateForPkcsSignedData()
DxeImageVerificationLibImageRead()
IsSignatureFoundInDatabase()
IsPkcsSignedDataVerifiedBySignatureList()
VerifyCertPkcsSignedData()

Authenticated Variables
gRT->GetVariable

MdePkg
BasePeCoffLib

PeCoffLoaderGetImageInfo()

CryptoPkg
BaseCryptLib

Sha256Init()
Sha256Update()
Sha256Final()
Sha256GetContextSize()

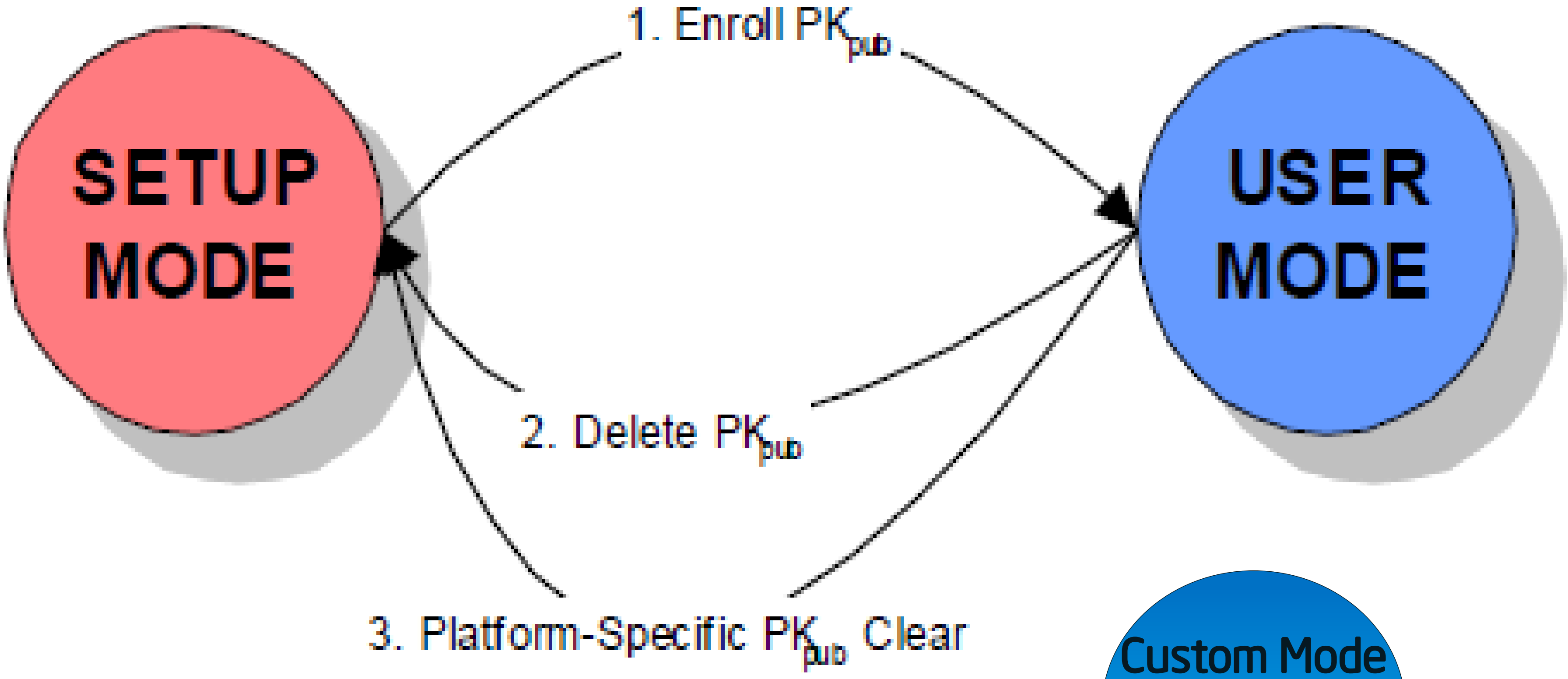
AuthenticodeVerify()
Pkcs7Verify()
WrapPkcs7Data()

OpenSslLib
Openssl-0.9.8w

IntrinsicLib

See *Rosenbaum, Zimmer, "A Tour Beyond BIOS into UEFI Secure Boot,*" for more details

Put them altogether: UEFI Secure Boot

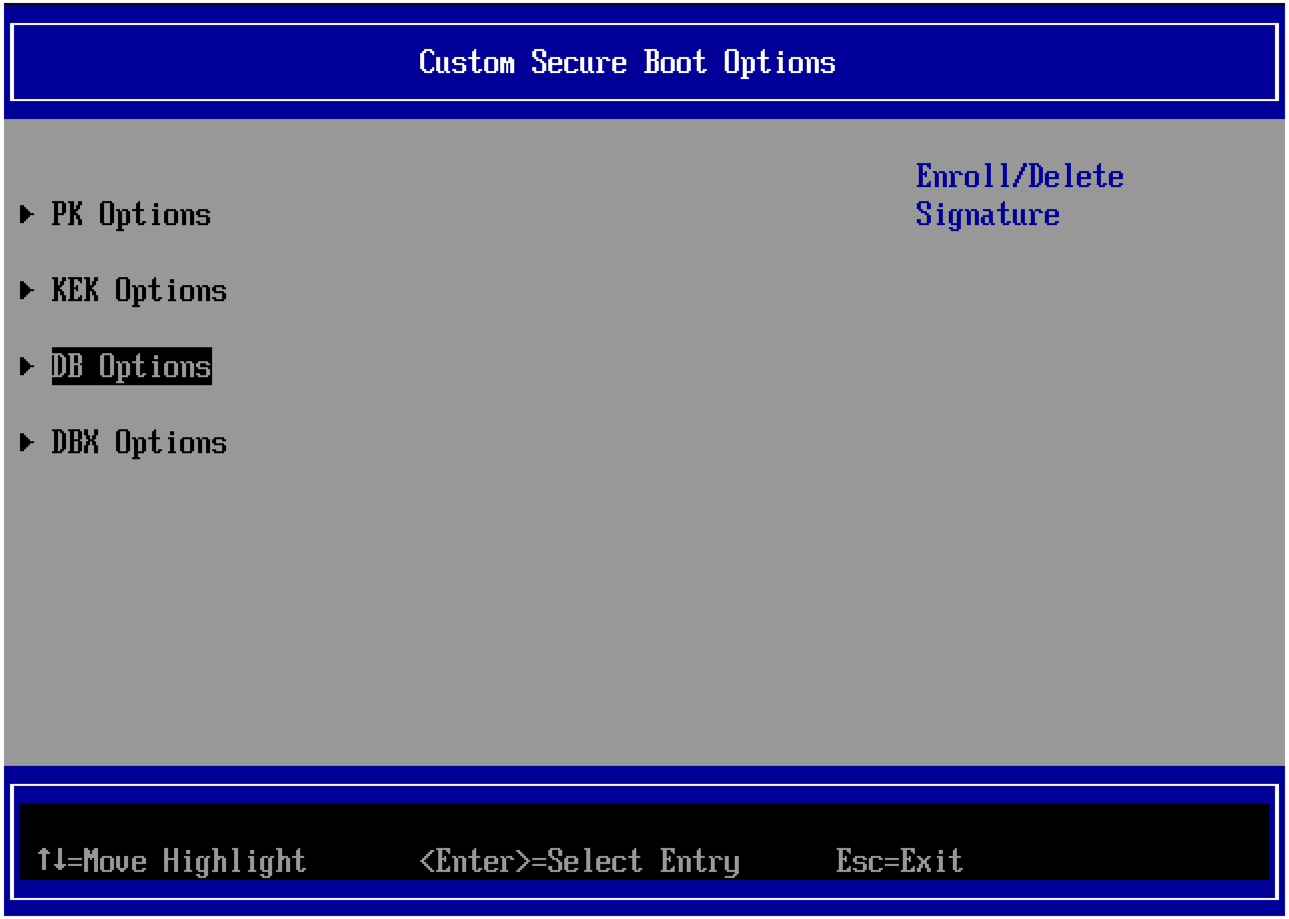


Custom Mode Edit keys on x86

Enable Secure Boot

End user controls -Custom Secure Boot Options

Enrolling DB and/or DBX for physically present user



Custom Secure Boot Options

▶ PK Options

▶ KEK Options

▶ DB Options

▶ DBX Options

Enroll/Delete Signature

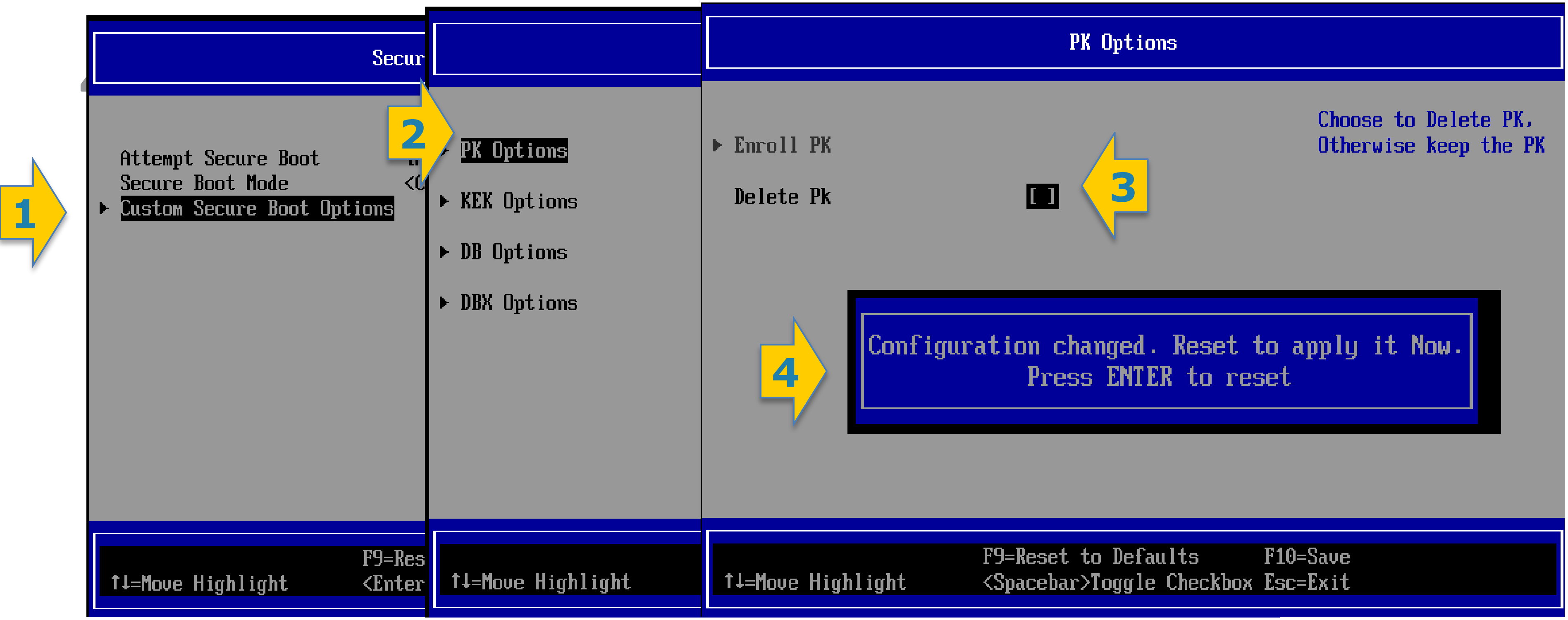
↑↓=Move Highlight   <Enter>=Select Entry   Esc=Exit

Disable Secure Boot

1. Select Custom Secure Boot Options

2. Select PK Options

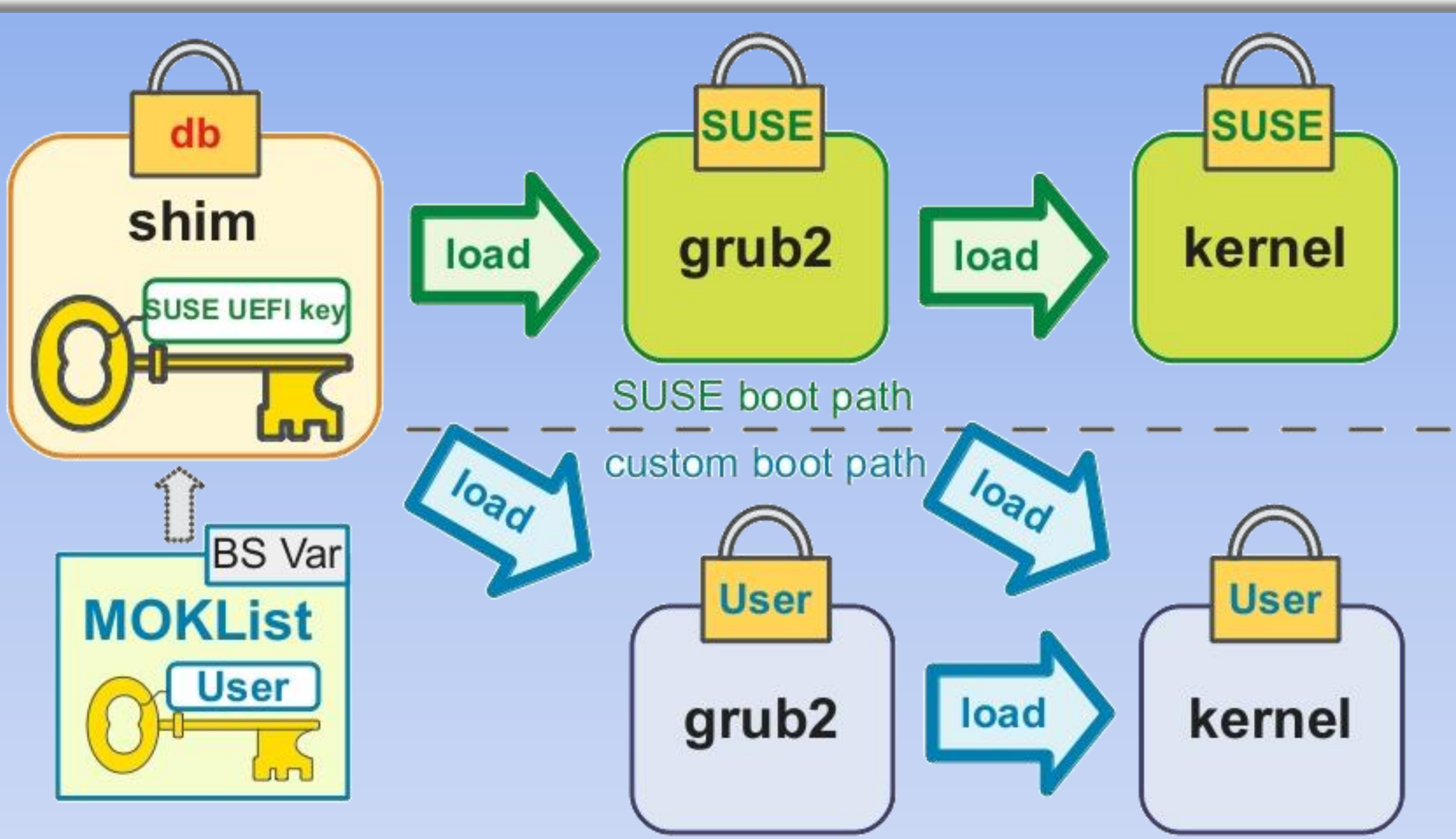
3. Delete Pk (space bar)



```
Secur

Attempt Secure Boot
Secure Boot Mode           <C
▶ Custom Secure Boot Options

                       F9=Res
↑↓=Move Highlight      <Enter
```

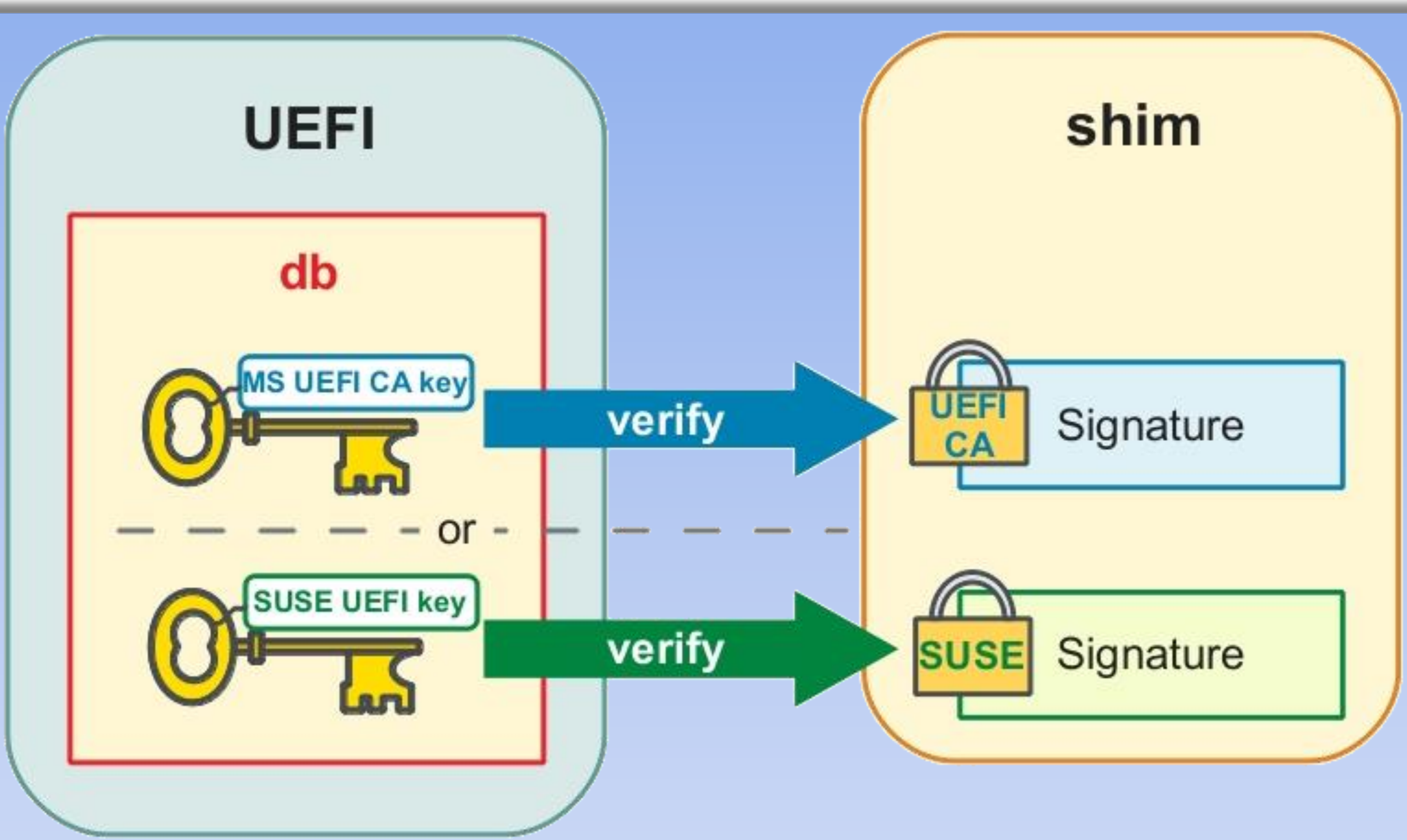
```
▶ PK Options
▶ KEK Options
▶ DB Options
▶ DBX Options

↑↓=Move Highlight
```

```
                              PK Options

▶ Enroll PK                                      Choose to Delete PK,
                                                 Otherwise keep the PK
  Delete Pk                [ ]

     Configuration changed. Reset to apply it Now.
              Press ENTER to reset

                      F9=Reset to Defaults      F10=Save
↑↓=Move Highlight     <Spacebar>Toggle Checkbox Esc=Exit
```



Load the UEFI image as long as it is trusted

Linux Update – Multiple OS Boot with MOK



Either the UEFI CA key or SUSE key will let the shim boot with UEFI secure boot

Multi-Signature Support for Shim

RandomNumberGenerator

UEFI driver implementing the EFI_RNG_PROTOCOL from the UEFI2.4 specification

TCG

PEI Modules & DXE drivers implementing Trusted Computing Group measured boot

EFI_TCG_PROTOCOL and EFI_TREE_PROTOCOL from the TCG and Microsoft MSDN websites, respectively

UserIdentification

DXE drivers that support multi-factor user authentication

Chapter 31 of the UEFI 2.4 specification

Library

DxeVerificationLib for "UEFI Secure Boot", chapter 27.2 of the UEFI 2.4 specification + other support libs

VariableAuthenticated

SMM and runtime DXE authenticated variable driver, chapter 7 of the UEFI2.4 specification

https://svn.code.sf.net/p/edk2/code/trunk/edk2/SecurityPkg

UDK2010 SecurityPkg

What to build & defend – Rationale for a threat model

“My house is secure” is almost meaningless

- Against a burglar? Against a meteor strike? A thermonuclear device?

“My system is secure” is almost meaningless

- Against what? To what extent?

Threat modeling is a process to define the goals and constraints of a (software) security solution

- Translate user requirements to security requirements

We used threat modeling for our UEFI / PI codebase

- We believe the process and findings are applicable to driver implementations as well as UEFI implementations in general

Defining, using a threat model

A Threat Model (TM) defines the security assertions and constraints for a product

- Assets: What we’re protecting
- Threats: What we’re protecting it against
- Mitigations: How we’re protecting our Assets

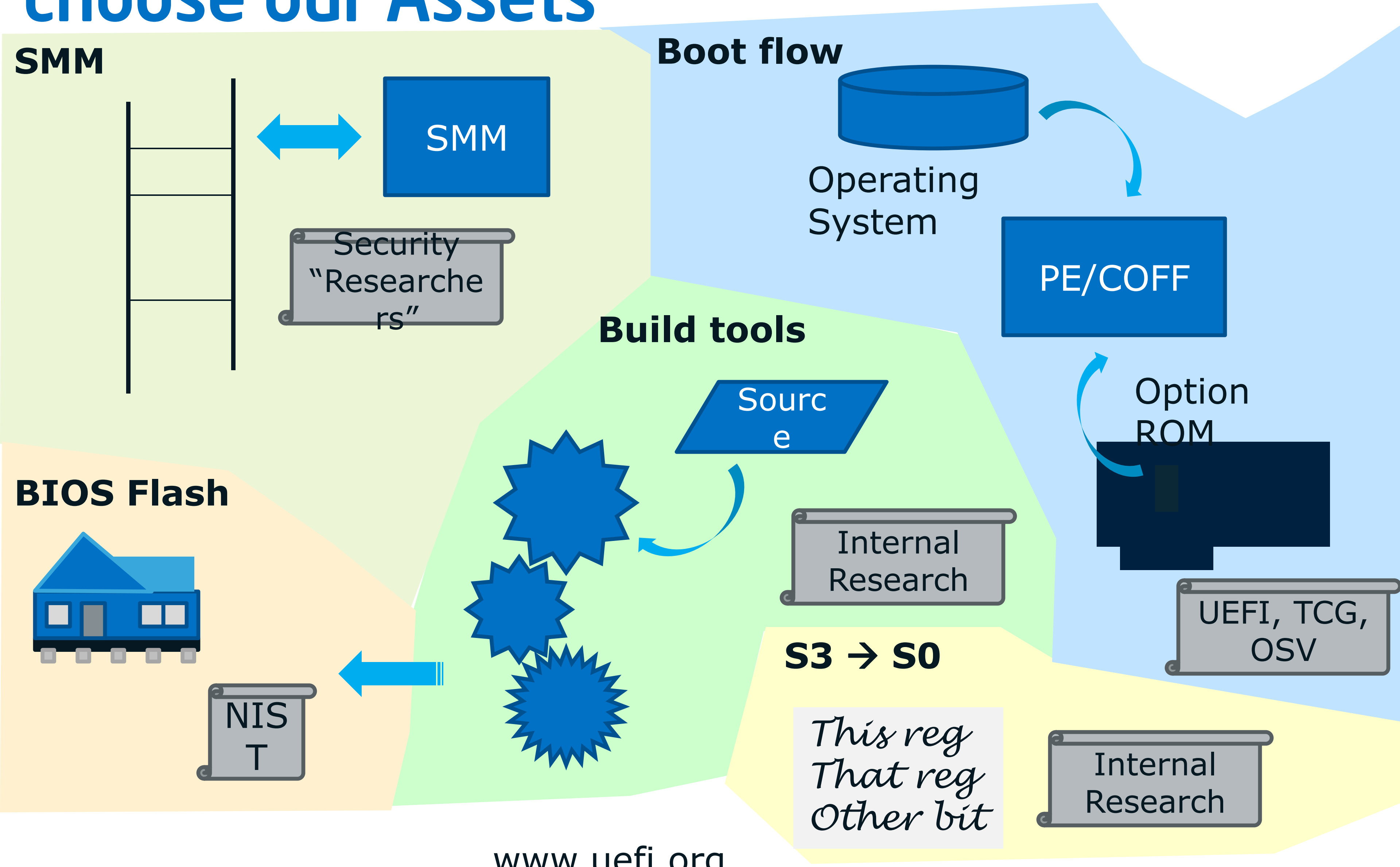
Use TM to narrow subsequent mitigation efforts

- Don’t secure review, fuzz test all interfaces
- Select the ones that are critical

TM is part science, part art, part experience, part nuance, part preference

- Few big assets vs lots of focused assets

We don’t always get to choose our Assets



SMM

SMM

Security “Researchers”

Boot flow

Operating System

PE/COFF

Option ROM

UEFI, TCG, OSV

Build tools

Source

Internal Research

BIOS Flash

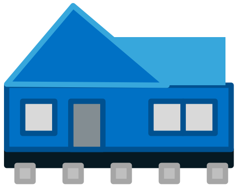
NIST

S3 → S0

This reg
That reg
Other bit

Internal Research

Flash**



NIST SP800-147 says

- Lock code flash except for update before Exit Mfg Auth
- Signed update (>= RSA2048, SHA256)
- High quality signing servers
- Without back doors (“non-bypassability”)

Threats

- PDOS – Permanent Denial of Service
  - System into inefficient room heater
- Elevation of privilege
  - Owning the system at boot is an advantage to a virus

Known attacks

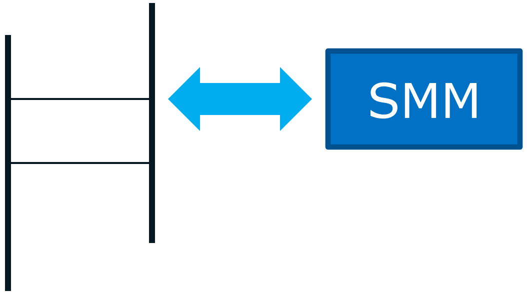
- CIH / Chernobyl 1999-2000
- Mebroni 2010

Mitigations include

- Reexamining flash protection methods – use the best even if its new
- Using advanced techniques to locate and remove (un)intentional backdoors

** or tomorrow’s equivalent NV storage

SMM



SMM is valuable because

- It’s invisible to Anti Virus, etc
- SMM sees all of system RAM
- Not too different from PCI adapter device firmware

Threats

- Elevation
  - View secrets or own the system by subverting RAM

Known attacks

- See e.g Duflot

Mitigations include

- Validate “external” / “untrusted” input
- Remove calls from inside SMM to outside SMM

Resume from S3

This reg
That reg
Other bit

ACPI says that we return the system to the S5→S0 configuration at S3→S0

- Must protect the data structures we record the cold boot config in

Threats

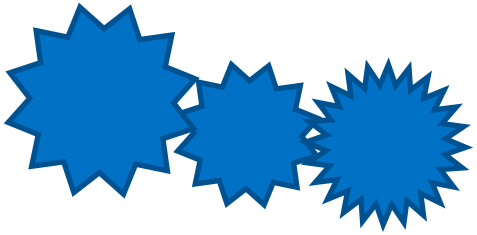
- Changing data structures could cause security settings to be incorrectly configured leaving S3
- Reopen the other assets' mitigated threats

No known attacks

Mitigations include

- Store data in SMM -or-
- Store hash of data structures and refuse to resume if the hashes don't compare

Tool chain



Tools create the resulting firmware

- Rely on third party tools and home grown tools
- Incorrect or attacked tools leave vulnerabilities

Threats

- Disabled signing, for example

Known attacks

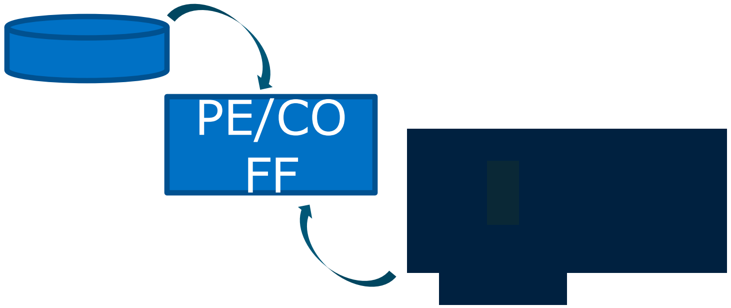
- See e.g. *Reflections on Trust*, Ken Thompson**

Mitigation

- Difficult: For most tools, provided as source code
- Review for correct implementation
- Use static, dynamic code analysis tools
  - PyLint for Python, for example

** CACM, Vol 27, No 8, Aug, 1984, pp. 761-763

Boot flow



Secure boot

- Authenticated variables
- Based on the fundamental Crypto being correct
- Correct location for config data

Threats

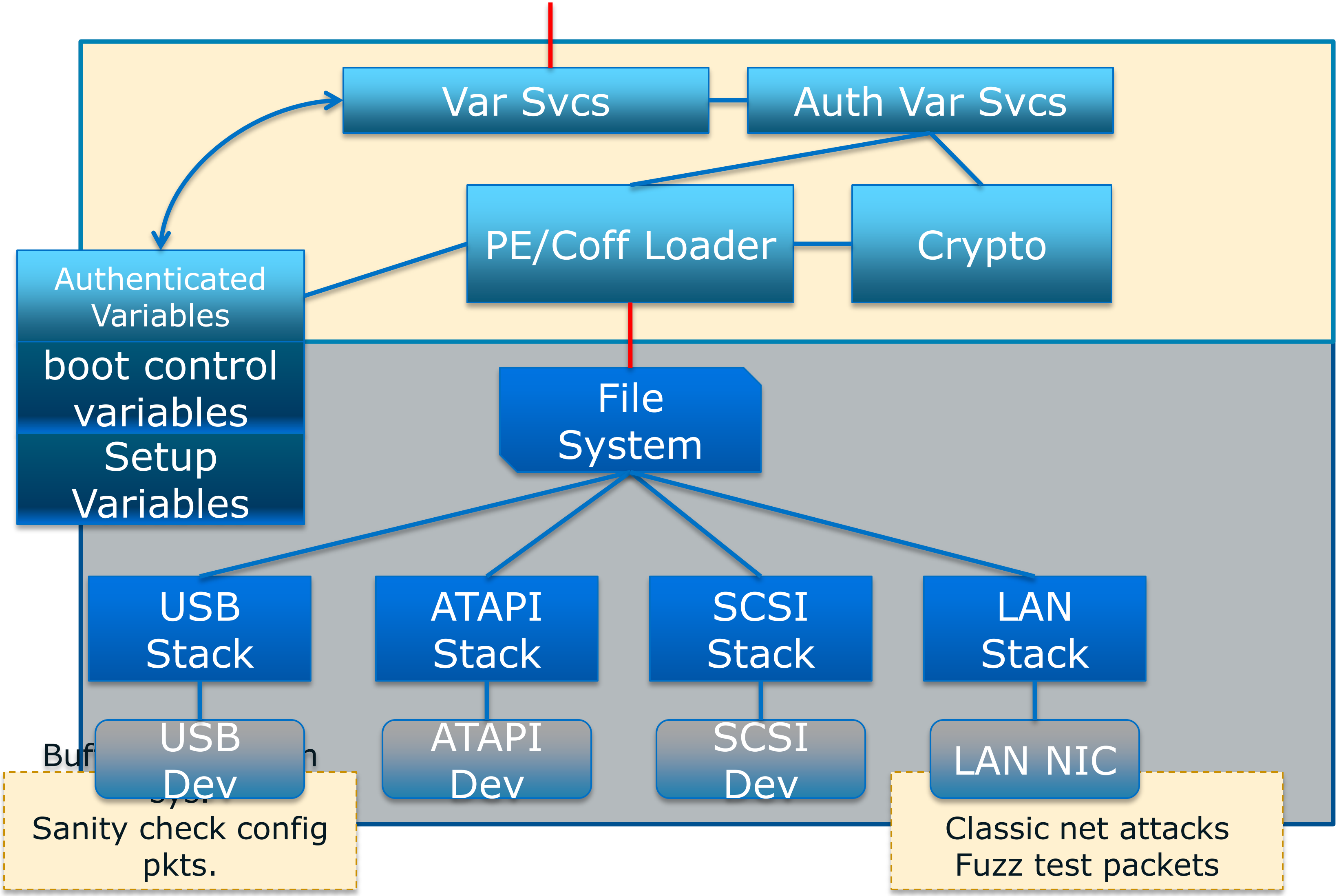
- Run unauthorized op roms, boot loaders
- PDOS systems with bad config variables

Known attacks

Mitigations include

- Sanity check config vars before use, use defaults
- Reviews, fuzz checking, third party reviews, etc.

TM to Modules: Boot flow



Buf n
sys.
Sanity check config pkts.

Classic net attacks
Fuzz test packets

Assets or not?

Variable content sanity checking?

- If you randomly fill in your Setup variables, will your system still boot?
- Fit in as a part of boot flow

ACPI? We create it but don't protect it

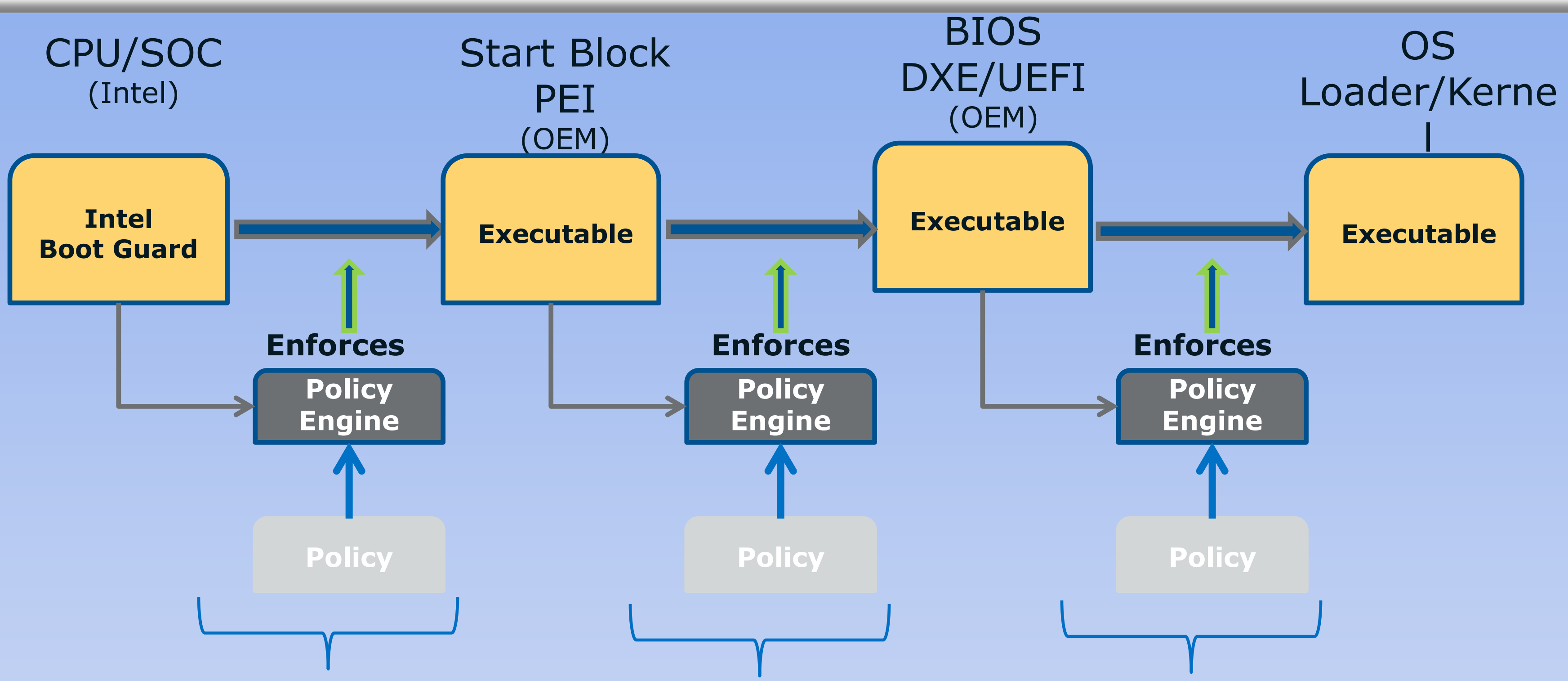
TPM support? We fill in the PCRs but don't use them (today)

Quality ≠ Security

Analyze and Mark external Interfaces where input can be attacker controlled data, comment headers

```
/**
  Install child handles if the Handle supports GPT partition structure.

  Caution: This function may receive untrusted input.
  The GPT partition table is external input, so this routine
  will do basic validation for GPT partition table before install
  child handle for each GPT partition.

  @param[in]  This       Calling context.
  @param[in]  Handle     Parent Handle.
  @param[in]  DevicePath Parent Device Path.

**/
EFI_STATUS
PartitionInstallGptChildHandl
```

UDK2010 example:
http://edk2.svn.sourceforge.net/svnroot/edk2/trunk/edk2/MdeModulePkg/Universal/Disk/PartitionDxe/Gpt.c



CPU/SOC
(Intel)

Start Block
PEI
(OEM)

BIOS
DXE/UEFI
(OEM)

OS
Loader/Kernel

Intel
Boot Guard

Executable

Executable

Executable

Enforces

Policy
Engine

Policy

Enforces

Policy
Engine

Policy

Enforces

Policy
Engine

Policy

Intel® Device Protection Technology with Boot Guard

http://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/4th-gen-core-family-mobile-brief.pdf

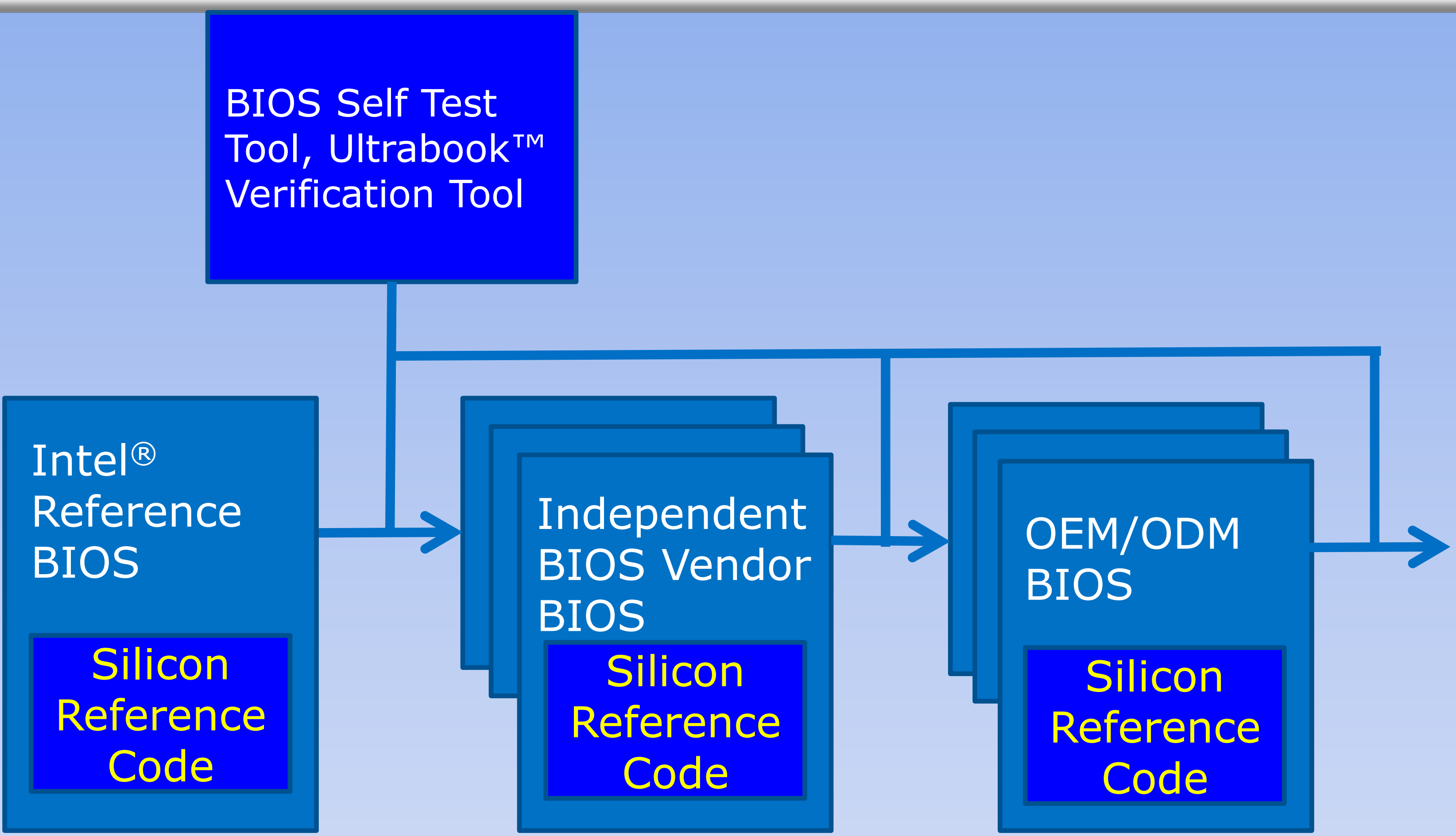
OEM PI Verification Using PI Signed Firmware Volumes

Vol 3, section 3.2.1.1
of PI 1.3 Specification

OEM UEFI 2.4 Secure Boot

Chapter 27.2 of
The UEFI 2.4
Specification

Intel® Boot Guard


BIOS Self Test Tool, Ultrabook™ Verification Tool

Intel® Reference BIOS

Silicon Reference Code

Independent BIOS Vendor BIOS

Silicon Reference Code

OEM/ODM BIOS

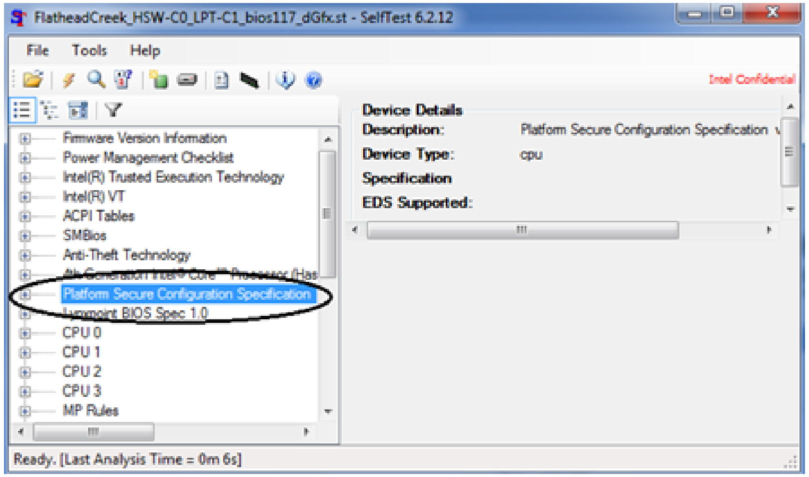
Silicon Reference Code

Suggested path to improve BIOS security assurance. Specific tests to verify SiRC are added in test tools.

Checking BIOS Security Compliance

SelfTest BIOS Validation

- Platform Secure Configuration Specification: Used to verify BIOS security

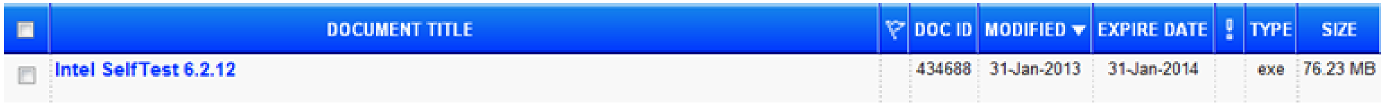

FlatheadCreek_HSW-C0_LPT-C1_bios117_dGfx.st - SelfTest 6.2.12

Device Details
Description: Platform Secure Configuration Specification v
Device Type: cpu
Specification
EDS Supported:

Ready. [Last Analysis Time = 0m 6s]

File
Tools
Help
Intel Confidential
Firmware Version Information
Power Management Checklist
Intel(R) Trusted Execution Technology
Intel(R) VT
ACPI Tables
SMBios
Anti-Theft Technology
4th Generation Intel® Core™ Processor (Has
Platform Secure Configuration Specification
Lynxpoint BIOS Spec 1.0
CPU 0
CPU 1
CPU 2
CPU 3
MP Rules

- Download SelfTest from CDI Doc# 434688
  http://www.intel.com/cd/edesign/library/asmo-na/eng/434688.htm



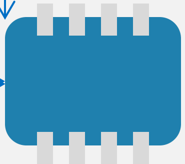
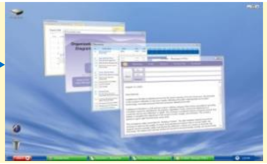
□	DOCUMENT TITLE		DOC ID	MODIFIED	EXPIRE DATE	!	TYPE	SIZE
□	Intel SelfTest 6.2.12		434688	31-Jan-2013	31-Jan-2014		exe	76.23 MB

SelfTest Checks BIOS Programming for Compliance

Platform Secure Configuration Spec/Tool

Technologies – putting it together

TCG Measurements into PCRs 0..7

Reset	Assets	Threats	
	BIOS Flash	ROM Swap Bit rot	H/W Spec
	**System BIOS** NIST SP800-147. Recovery. DXE SMM, UEFI Core	Erase flash part Overwrite flash part	SP800-147
	Option ROMs BIOS device drivers	Erase op ROM Overwrite op ROM	UEFI 2.4
	Network Boot IPv6 for the cloud	Network attacks	
	**OS Boot loader** BIOS loads the OS To BIOS, Hv/VMM is an OS	Spoof boot loader	

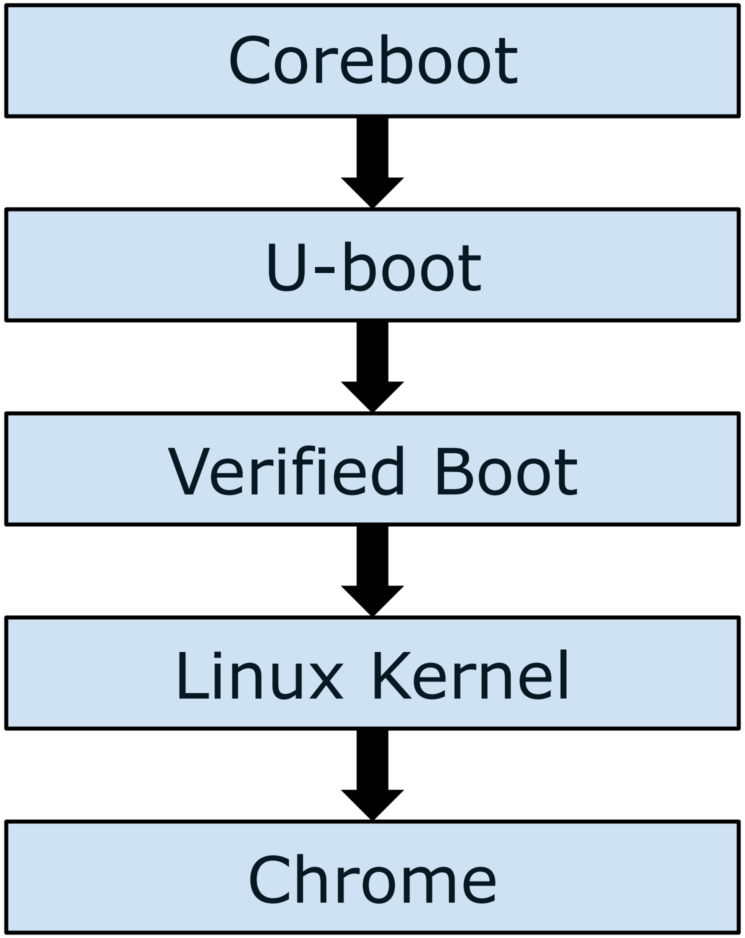
Different colors for different vendors

Just UEFI/EDK2? Also Intel booting via Coreboot for Chromebooks

Open

- GPLv2
- Mostly written in C
- Kconfig and modified Kbuild
- High-level organization not too different from EFI
  - Well-defined boot phases
  - Modular CPU, Chipset, Device support
- NOT a bootloader
  - Support for various payloads
  - Payloads can boot Linux, DOS, Windows, etc

Basic Coreboot Boot Flow



Coreboot

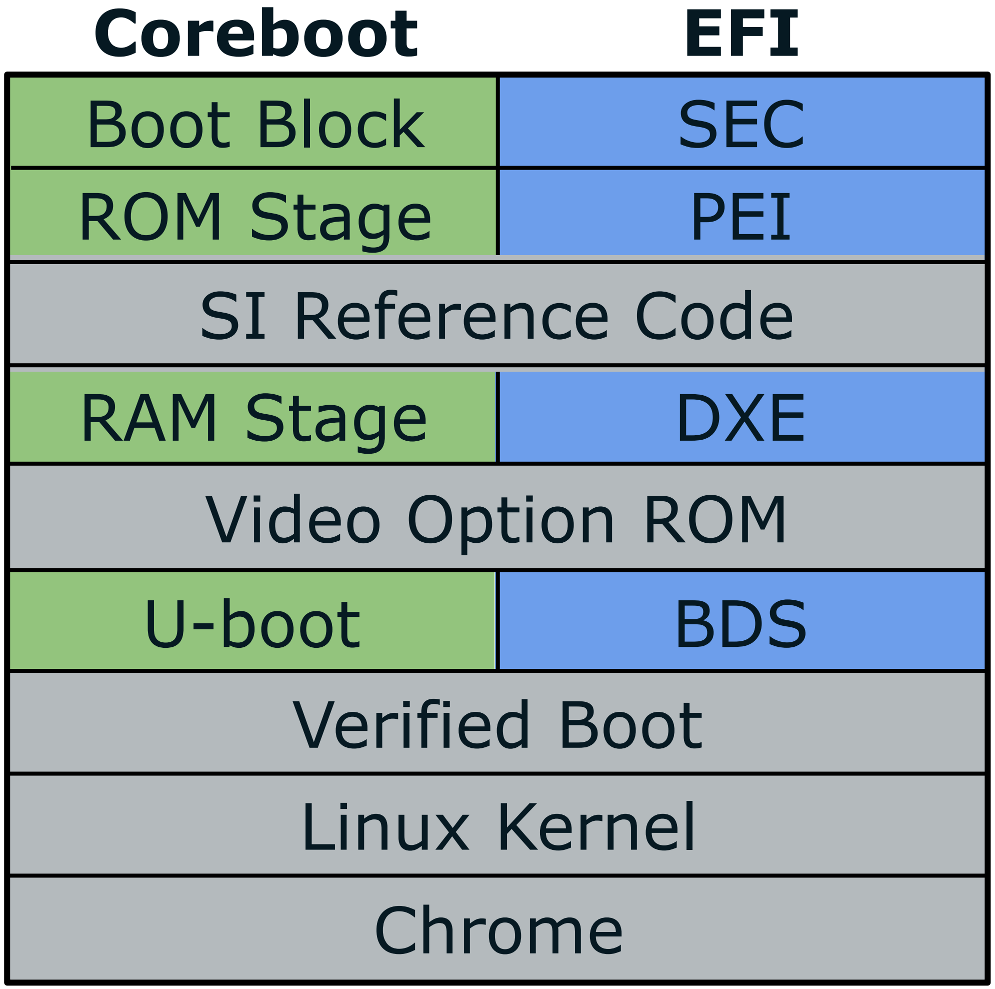
U-boot

Verified Boot

Linux Kernel

Chrome

Coreboot vs. UEFI

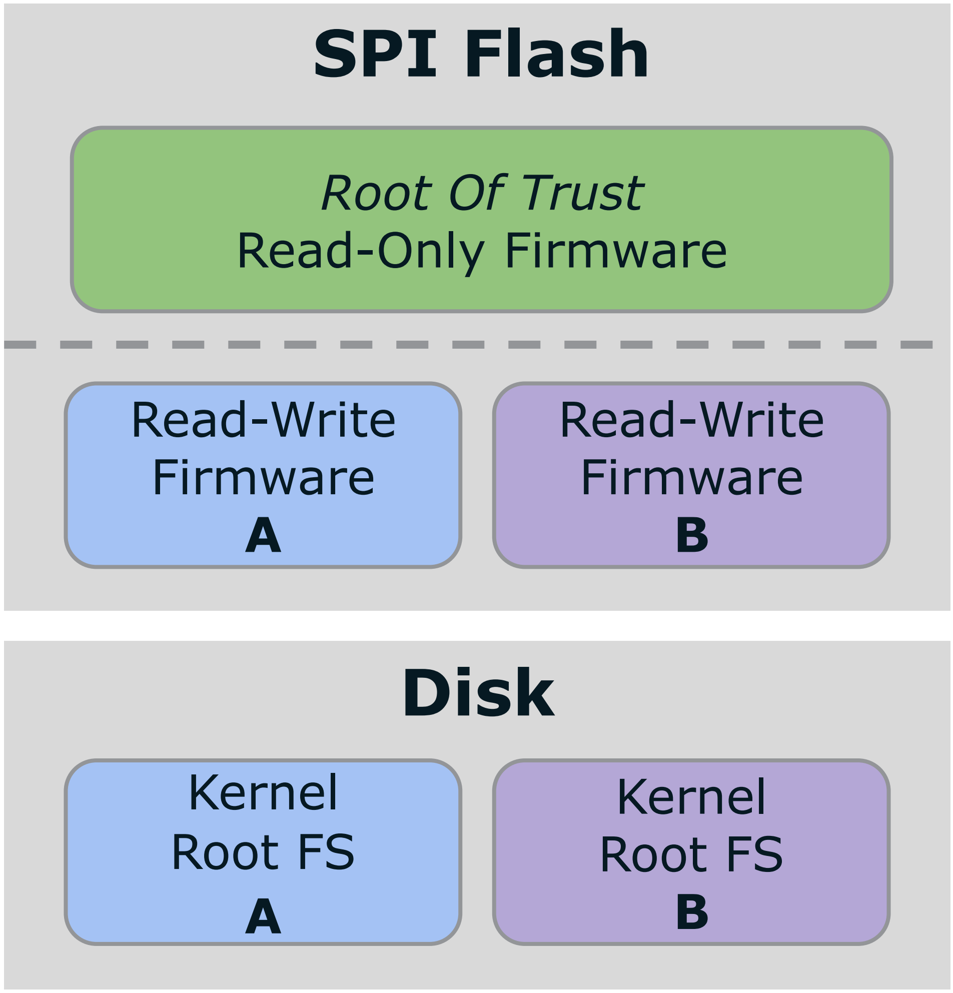


<table>
<tr><th>Coreboot</th><th>EFI</th></tr>
<tr><td>Boot Block</td><td>SEC</td></tr>
<tr><td>ROM Stage</td><td>PEI</td></tr>
<tr><td colspan="2">SI Reference Code</td></tr>
<tr><td>RAM Stage</td><td>DXE</td></tr>
<tr><td colspan="2">Video Option ROM</td></tr>
<tr><td>U-boot</td><td>BDS</td></tr>
<tr><td colspan="2">Verified Boot</td></tr>
<tr><td colspan="2">Linux Kernel</td></tr>
<tr><td colspan="2">Chrome</td></tr>
</table>

Verified Boot - Firmware

- Root Of Trust is in read-only firmware
  - Reset vector must be in RO flash
  - Complicated by SPI Flash Descriptor and ME
- RO firmware can verify signed RW firmware
- Firmware verifies signed kernel from disk
- Reference implementation available
  - chromiumos/platform/vboot_reference.git

Verified Boot - Overview



SPI Flash

Root Of Trust
Read-Only Firmware

Read-Write
Firmware
A

Read-Write
Firmware
B

Disk

Kernel
Root FS
A

Kernel
Root FS
B

Challenges

- Multi-OS support, GPL3 & Open source, binary + source
- Firmware size – open source & crypto libs
- Speed impacts
- Consistency w/ other ‘security’ technologies in platform
- Robustness
  - Coding practice
  - Protected updates
  - Recovery
- Validation
  - Negative testing
  - Fuzzing
- Agreement on threat model across ecosystem
- Disclosure, response, fix cycle
- Updates
- Interoperability of different implementations

Summary

- Threats of firmware attacks & UEFI extensibility are real
- Address w/ open standards and open source
- Secure boot is coming w/ next OS wave (and like longevity of any shrinkwrap OS release, will continue for 10 yrs)
- Challenges in ecosystem enabling

For more information - UEFI Secure Boot

Intel Technology Journal, Volume 15, Issue 1, 2011, UEFI Today: Bootstrapping the Continuum, UEFI Networking and Pre-OS Security, page 80 at http://www.intel.com/technology/itj/2011/v15i1/pdfs/Intel-Technology-Journal-Volume-15-Issue-1-2011.pdf

Rosenbaum, Zimmer, "A Tour Beyond BIOS into UEFI Secure Boot," Intel Corporation, July 2012 http://sourceforge.net/projects/edk2/files/General%20Documentation/A_Tour_Beyond_BIOS_into_UEFI_Secure_Boot_White_Paper.pdf/download

UEFI 2.3.1 specification: Sections 7.2 (Variable Services) and Sections 27.2 through 27.8 (Secure Boot) of the at www.uefi.org

Beyond BIOS: Developing with the Unified Extensible Firmware Interface, 2nd Edition, Zimmer, et al, ISBN 13 978-1-934053-29-4, Chapter 10 – Platform Security and Trust, http://www.intel.com/intelpress

“Hardening the Attack Surfaces,” MSFT 2012 UEFI Plugfest http://www.uefi.org/learning_center/UEFI_Plugfest_2012Q1_Microsoft_AttackSurface.pdf

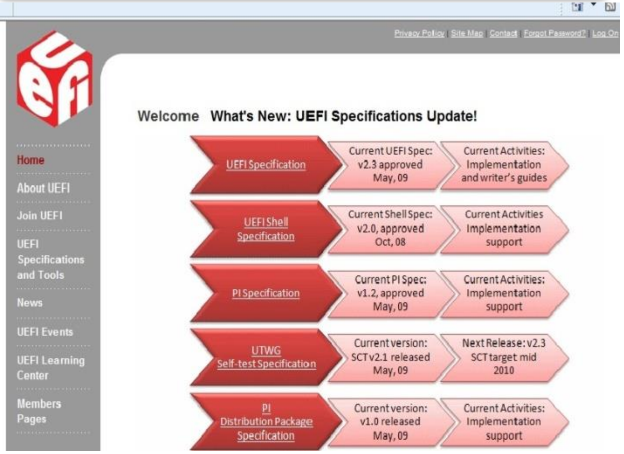
“Building hardware-based security with a TPM” MSFT BUILD http://channel9.msdn.com/Events/BUILD/BUILD2011/HW-462T

Lin, Oswald, Zimmer, “UEFI Secure Boot in Linux,” Intel Developer Forum, San Francisco, September 11, 2013 https://intel.activeevents.com/sf13/connect/fileDownload/session/A25811835C1B6573651FC73FB20D0F6C/SF13_STTS002_100.pdf

A Tale of One Software Bypass of Windows 8 Secure Boot by Andrew Furtak, Oleksandr Bazhaniuk and Yuriy Bulygin, Blackhat 2013

UEFI Industry Resources

UEFI Forum



Privacy Policy | Site Map | Contact | Forgot Password? | Log On

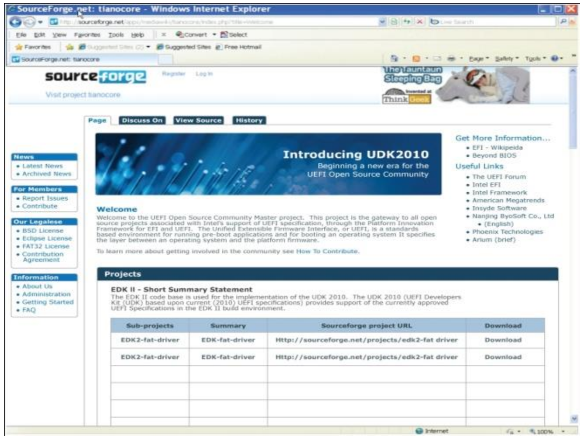
Home
About UEFI
Join UEFI
UEFI Specifications and Tools
News
UEFI Events
UEFI Learning Center
Members Pages

Welcome **What's New: UEFI Specifications Update!**

UEFI Specification	Current UEFI Spec: v2.3 approved May, 09	Current Activities: Implementation and writer's guides
UEFI Shell Specification	Current Shell Spec: v2.0, approved Oct, 08	Current Activities Implementation support
PI Specification	Current PI Spec: v1.2, approved May, 09	Current Activities: Implementation support
UTWG Self-test Specification	Current version: SCT v2.1 released May, 09	Next Release: v2.3 SCT target mid 2010
PI Distribution Package Specification	Current version: v1.0 released May, 09	Current Activities: Implementation support

www.uefi.org

UEFI Open Source



SourceForge.net: tianocore - Windows Internet Explorer

News
- Latest News
- Archived News

For Members
- Report Issues
- Contribute

Our Legalese
- BSD License
- Eclipse License
- FAT32 License
- Contribution Agreement

Information
- About Us
- Administration
- Getting Started
- FAQ

Introducing UDK2010
Beginning a new era for the
UEFI Open Source Community

Welcome

Welcome to the UEFI Open Source Community Master project. This project is the gateway to all open source projects associated with Intel's support of UEFI specification, through the Platform Innovation Framework for EFI and UEFI. The Unified Extensible Firmware Interface, or UEFI, is a standards based environment for running pre-boot applications and for booting an operating system It specifies the layer between an operating system and the platform firmware.

To learn more about getting involved in the community see How To Contribute.

Get More Information...
- EFI - Wikipeida
- Beyond BIOS

Useful Links
- The UEFI Forum
- Intel EFI
- Intel Framework
- American Megatrends
- Insyde Software
- Nanjing ByoSoft Co., Ltd
  - (English)
- Phoenix Technologies
- Arium (brief)

Projects

EDK II - Short Summary Statement

The EDK II code base is used for the implementation of the UDK 2010. The UDK 2010 (UEFI Developers Kit (UDK) based upon current (2010) UEFI specifications) provides support of the currently approved UEFI Specifications in the EDK II build environment.

Sub-projects	Summary	Sourceforge project URL	Download
EDK2-fat-driver	EDK-fat-driver	Http://sourceforge.net/projects/edk2-fat driver	Download
EDK2-fat-driver	EDK-fat-driver	Http://sourceforge.net/projects/edk2-fat driver	Download

www.tianocore.org

Intel UEFI Resources

Intel® UDK Community Resource Center

Welcome to Intel® UDK Community Resource Center

Learn more about UEFI ›

Learn.

Communicate.

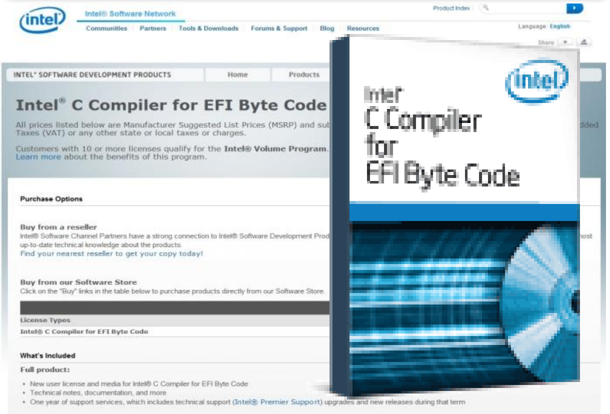
Collaborate.

Develop.

Integrate.

www.intel.com/UDK

Intel EBC Compiler



Intel® C Compiler for EFI Byte Code

All prices listed below are Manufacturer Suggested List Prices (MSRP) and sub... Taxes (VAT) or any other state or local taxes or charges.

Customers with 10 or more licenses qualify for the **Intel® Volume Program**. Learn more about the benefits of this program.

Purchase Options

Buy from a reseller
Intel® Software Channel Partners have a strong connection to Intel® Software Development Prod... up-to-date technical knowledge about the products.
Find your nearest reseller to get your copy today!

Buy from our Software Store
Click on the "Buy" links in the table below to purchase products directly from our Software Store.

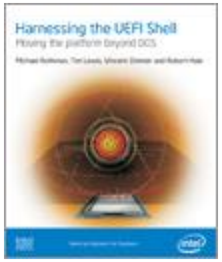
License Types
Intel® C Compiler for EFI Byte Code

What's Included

Full product:
- New user license and media for Intel® C Compiler for EFI Byte Code
- Technical notes, documentation, and more
- One year of support services, which includes technical support (Intel® Premier Support) upgrades and new releases during that term

http://software.intel.com/en-us/articles/intel-c-compiler-for-efi-byte-code-purchase/

UEFI Books/ Collateral



Harnessing the UEFI Shell
Moving the platform beyond DOS

Beyond BIOS:
Developing with the Unified Extensible Firmware Interface

Intel® Technology Journal
UEFI Today:
Bootstrapping the Continuum

www.intel.com/intelpress

http://www.intel.com/technology/itj/2011/v15i1/index.htm

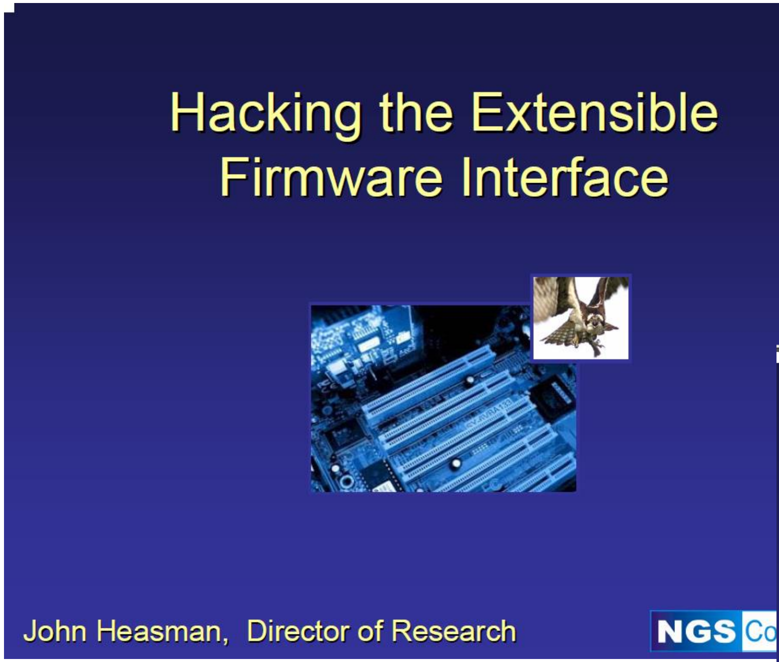
BSidesSeattle

Thank You

Contact:
vincent.zimmer@gmail.com
@vincentzimmer

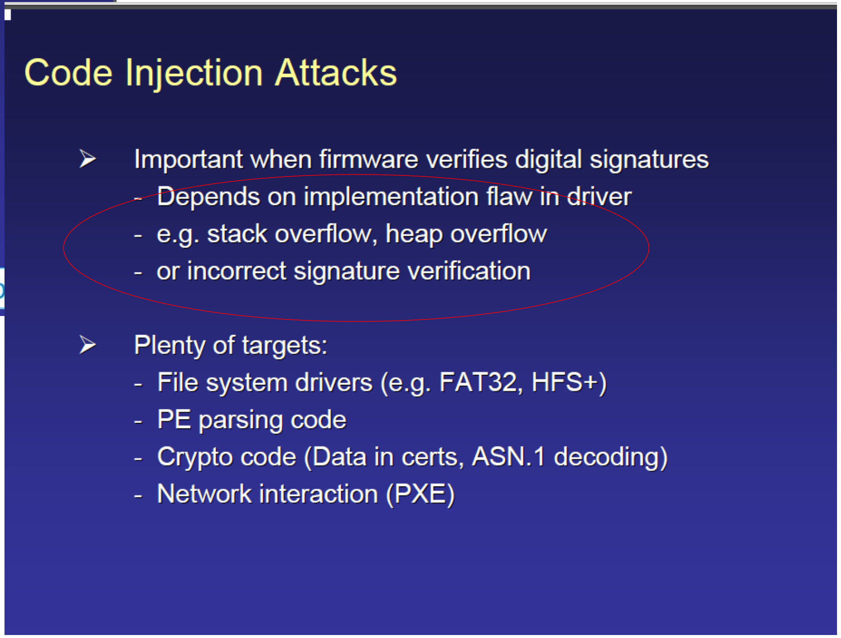
Backup

History of attacks – 2007 – Blackhat Las Vegas



Hacking the Extensible Firmware Interface

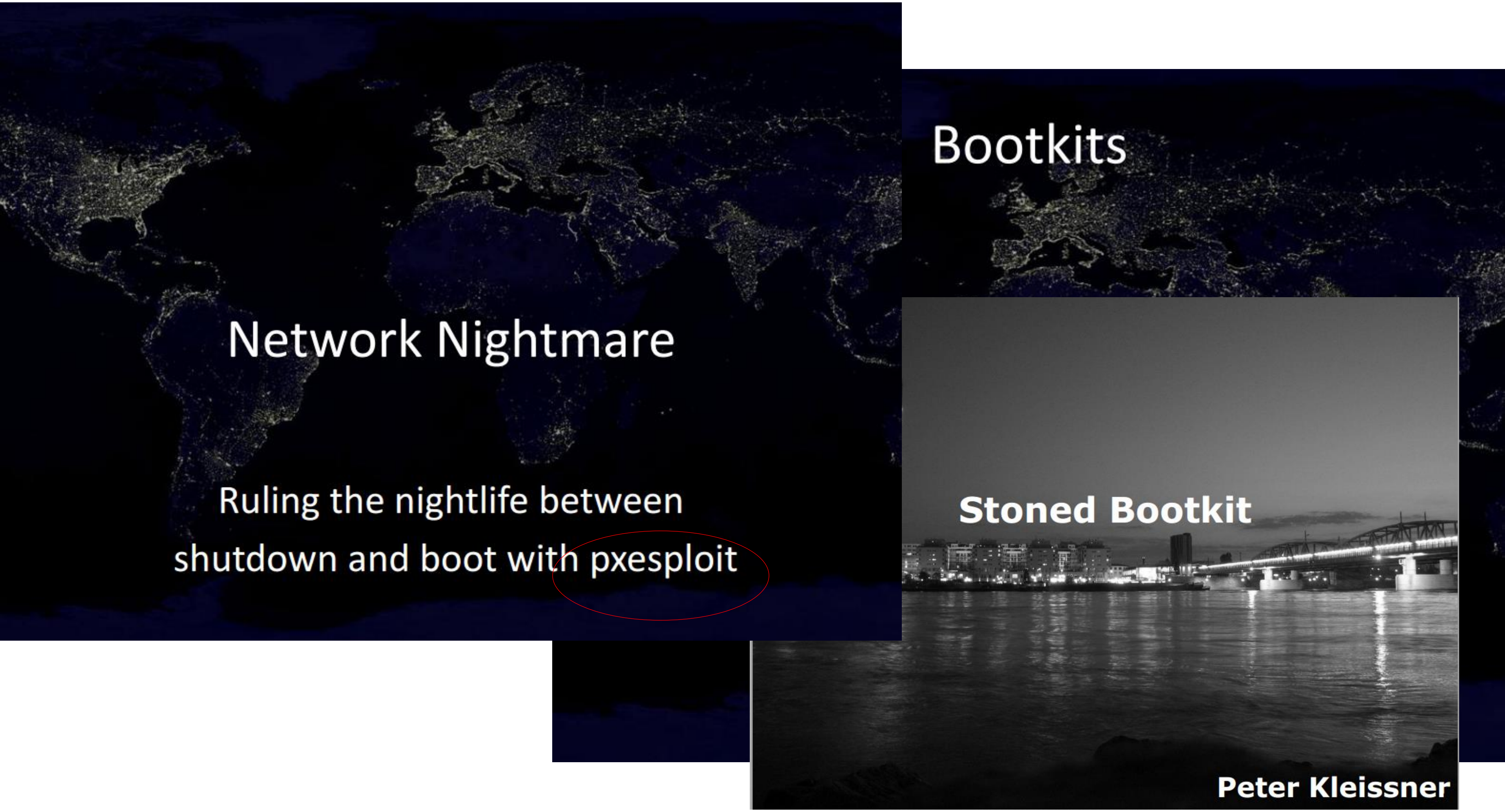
John Heasman, Director of Research



Code Injection Attacks

- Important when firmware verifies digital signatures
  - Depends on implementation flaw in driver
  - e.g. stack overflow, heap overflow
  - or incorrect signature verification
- Plenty of targets:
  - File system drivers (e.g. FAT32, HFS+)
  - PE parsing code
  - Crypto code (Data in certs, ASN.1 decoding)
  - Network interaction (PXE)

Defcon 19 – Bootkits and network boot attacks



Network Nightmare

Ruling the nightlife between shutdown and boot with pxesploit

Bootkits

Stoned Bootkit

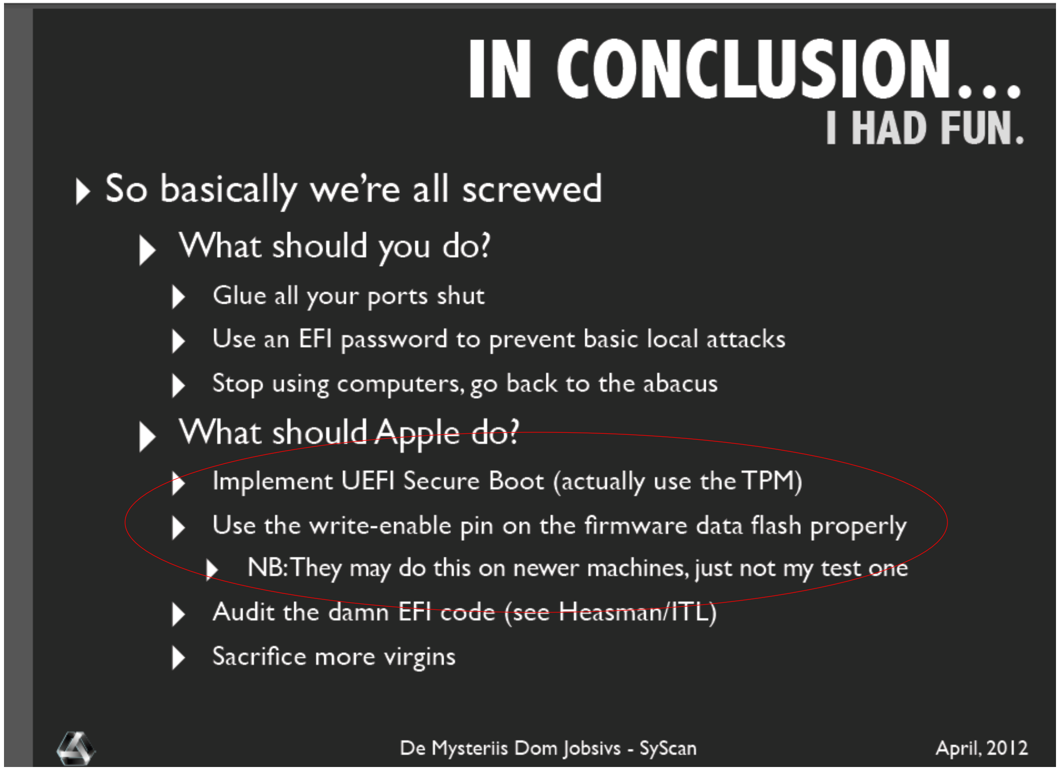
Peter Kleissner

SYSCAN Singapore – April 2012

DE MYSTERIIS DOM JOBSIVS: MAC EFI ROOTKITS

SNARE
@ SYSCAN SINGAPORE
APRIL 2012

assurance



IN CONCLUSION…

I HAD FUN.

- So basically we're all screwed
  - What should you do?
    - Glue all your ports shut
    - Use an EFI password to prevent basic local attacks
    - Stop using computers, go back to the abacus
  - What should Apple do?
    - Implement UEFI Secure Boot (actually use the TPM)
    - Use the write-enable pin on the firmware data flash properly
      - NB: They may do this on newer machines, just not my test one
    - Audit the damn EFI code (see Heasman/ITL)
    - Sacrifice more virgins

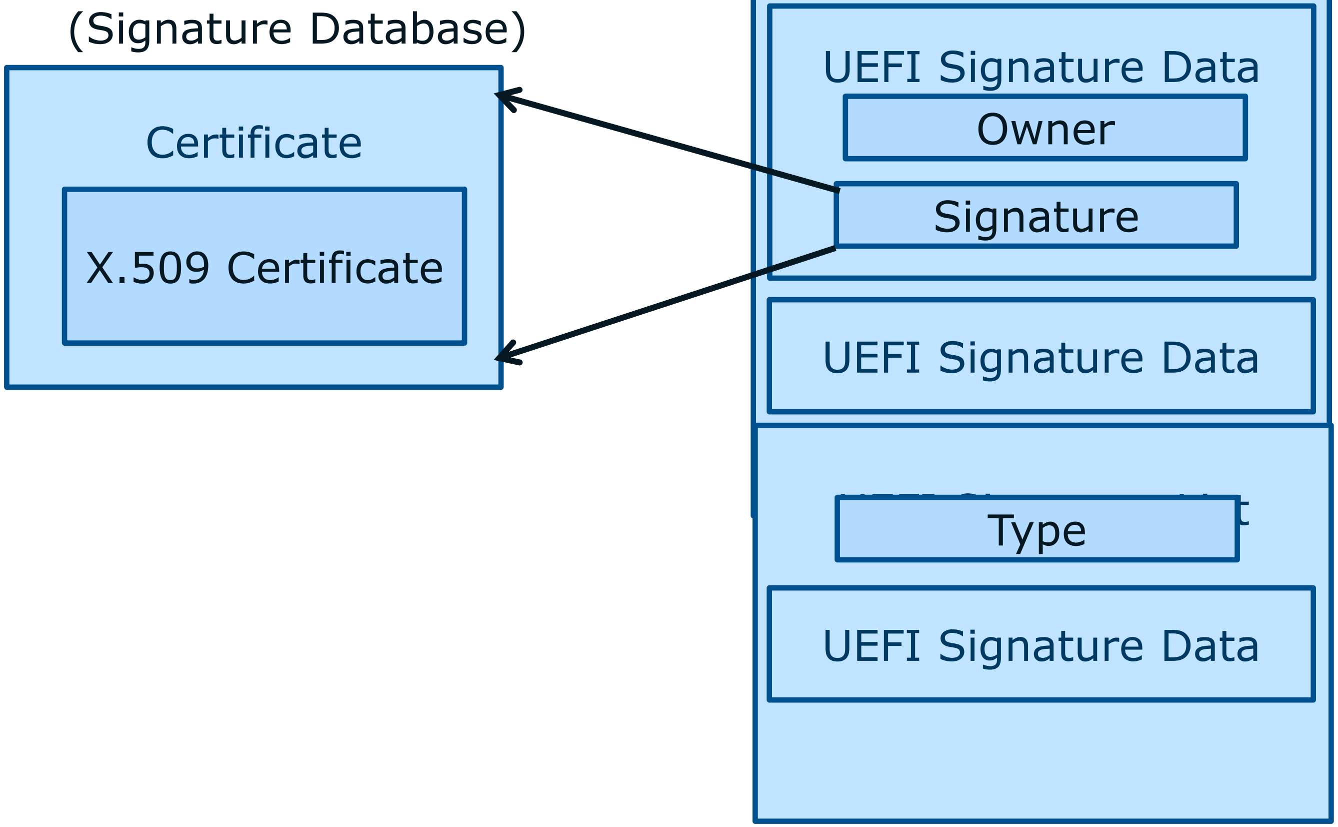
De Mysteriis Dom Jobsivs - SyScan

April, 2012

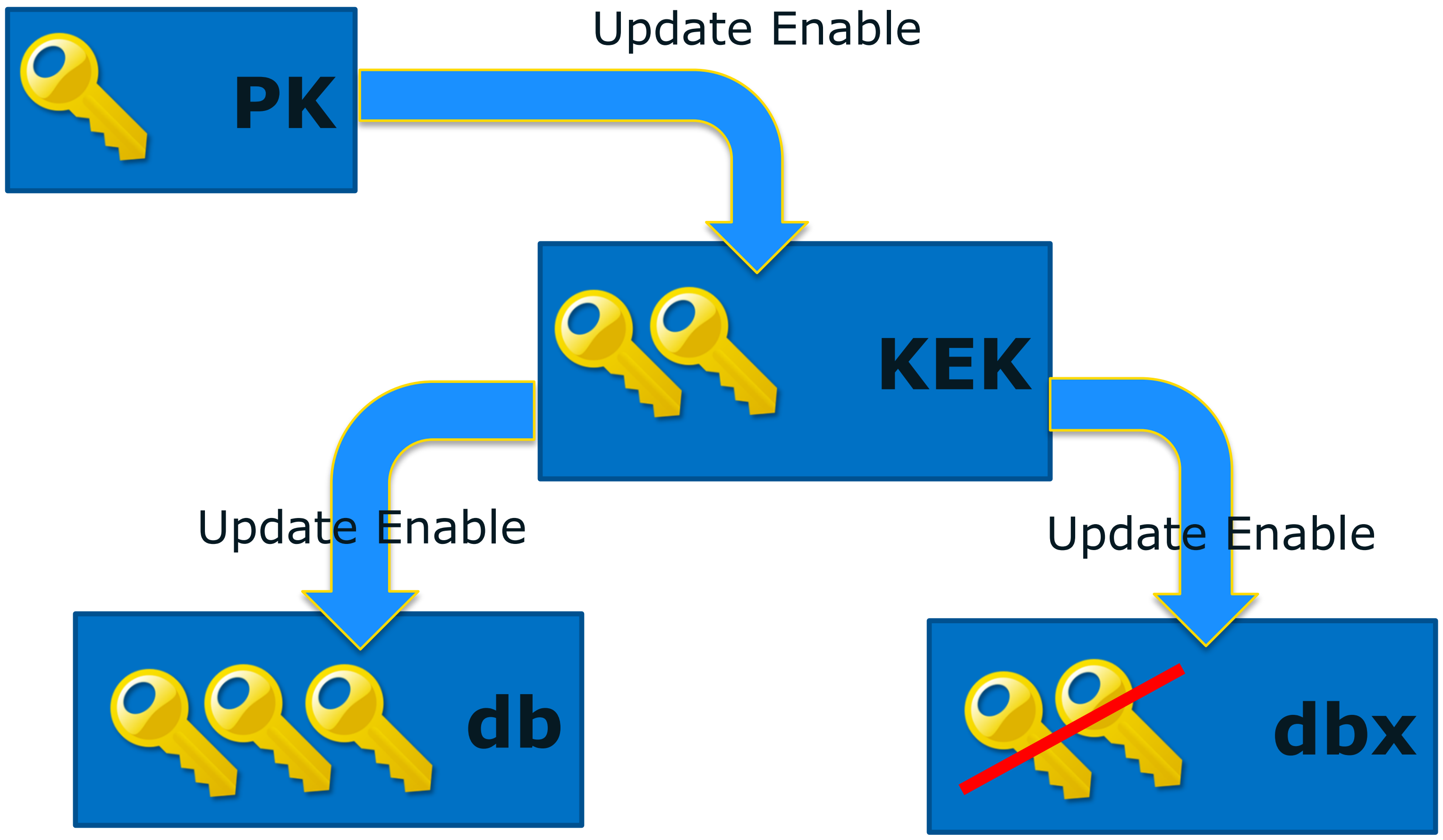
Firmware/OS Key

Why? – How can firmware know if certificate is valid?

How? – Firmware/OS Key



UEFI Secure Boot Database Review



If Signed by key in db, driver or loader can Run!

If Signed by key in dbx, driver/loader forbidden!

Who "Owns" The System Security Keys?

PK – Key pair is created by Platform Manufacturer

Typically one PK pair used for a model or model Line

KEK – Key supplied by OS Partner,

Optional: Include 2nd key created by OEM

db – OS vendor supplies Key,

CA supplies Key,

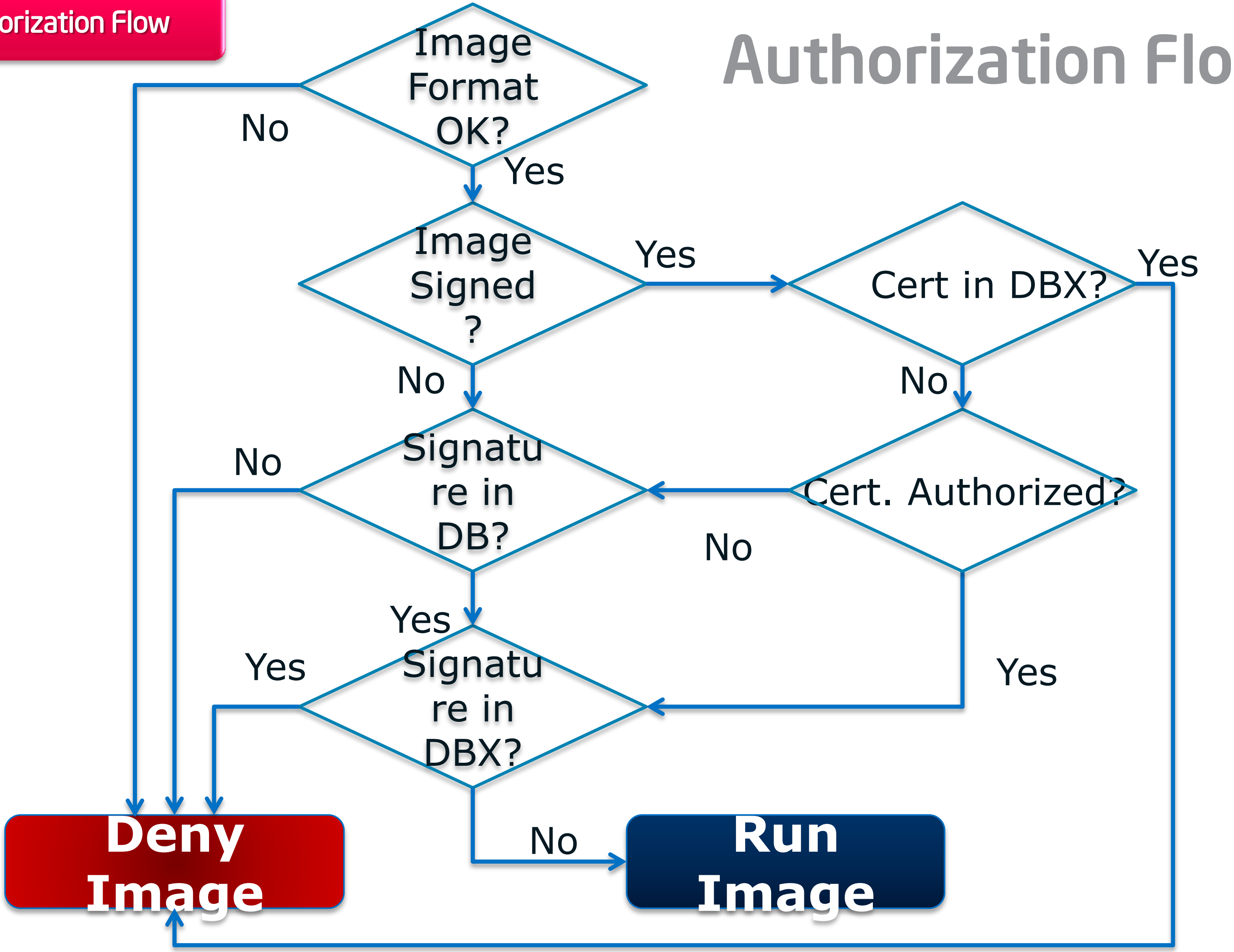
Optional: OEM App Signing Key

dbx – list of revoked keys

- Signing authority issues revoked keys

Signature Tests using db Keys Block Rogue S/W!

Authorization Flow



See *Rosenbaum, Zimmer, "A Tour Beyond BIOS into UEFI Secure Boot,"* for more details