UEFI Tutorial
8/29/12
Harry Hsiung
Intel Corp
Agenda

• UEFI Basics
• UEFI security features
• UEFI development platforms
• UEFI Resources
• Opens
• Backup
UEFI architecture

OPERATING SYSTEM

Legacy OS LOADER

UEFI OS LOADER

UEFI BOOT SERVICES

UEFI or PI Drivers

Protocols + Handlers

Memory

Timer

Platform Specific Firmware

Platform Hardware

Motherboard ROM/FLASH Firmware vol

Option ROM

UEFI Drivers

UEFI SYSTEM PARTITION

OS PARTITION

UEFI API

UEFI RUNTIME SERVICES

UEFI architecture

SMBIOS

ACPI

INTERFACES FROM OTHER REQUIRED SPECS

(OTHER)

Compatibility

Plumbers conference 2012
Plumbers conference 2012

**UEFI Specification Timeline**

- **UEFI 2.0**
- **UEFI 2.1**
- **UEFI 2.2**
- **UEFI 2.3**
- **UEFI 2.3.1**

**Specifications**

- PI 1.0
- PI 1.1
- PI 1.2
- Shell 2.0
- Packaging 1.0

**Years**

- 2006
- 2007
- 2008
- 2009
- 2010
- 2011

**Implementation**

- **EDK 1.01: UEFI 2.0**
- **EDK 1.04: UEFI 2.1**
- **EDK 1.06: UEFI 2.1+**
- **EDK II*: UEFI 2.1+**
- **UDK2010: UEFI 2.3+**
- **UDK2010.S R1: UEFI 2.3.1+**

**Open Source code Tianocore.org**

* EDK II is same code base as UDK2010  
* EDK I is UEFI 2.0..UEFI 2.1(1117)

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

* EDK II is same code base as UDK2010
Welcome  What's New: UEFI Specifications Update!

- **UEFI Specification**
  - Current UEFI Spec: v2.3.1 approved April 8, 2011
  - 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 v2.3 released Jan. 11
  - Next Release: v2.3.1 SCT target April 2012

- **PI Distribution Package Specification**
  - Current version: v1.0 released May, 09
  - Current Activities: Implementation support
Architecture Execution Flow

<table>
<thead>
<tr>
<th>Security (SEC)</th>
<th>Pre EFI Initialization (PEI)</th>
<th>Driver Execution Environment (DXE)</th>
<th>Boot Dev Select (BDS)</th>
<th>Transient System Load (TSL)</th>
<th>Run Time (RT)</th>
<th>After Life (AL)</th>
</tr>
</thead>
</table>

Power on [Platform initialization ...]

[OS boot ...]  
Shutdown

Boot Execution Flow
DXE Foundation Data Structures

**Active Consoles**
- Input Console
- Output Console
- Standard Error Console

**UEFI Boot Services Table**
- Task Priority Level Services
- Memory Services
- Event and Timer Services
- Protocol Handler Services
- Image Services
- Driver Support Services

**UEFI Runtime Services Table**
- Variable Services
- Real Time Clock Services
- Reset Services
- Status Code Services
- Virtual Memory Services

**Version Information**
- UEFI Specification Version
- Firmware Vendor
- Firmware Revision

**DXE Services Table**
- Global Coherency Domain Services
- Dispatcher Services

**System Configuration Table**
- DXE Services Table
- HOB List
- ACPI Table
- SMBIOS Table
- ...
- SAL System Table

**Handle Database**

**Protocol Interface**

**Boot Service Data Structures**

**Runtime Data Structures**
Boot Support - Device Types

- Hard disk
- Raid
- Fiber channel
- Removable media
  - CD-ROM, DVD-ROM
    - El Torito 1.0 “No emulation”
    - Floppy, USB Storage, etc.
- Network
  - PXE BIOS support specification (Wire for Management)
  - iSCSI
- Future media via extensibility methods
GPT - New Partition Structure

Boot Support

First useable block

LBA0 LBA1

MBR

Partition Table HDR

0 1 ... n

Start partition

End partition

Partition 1

Last useable block

0 1 ... n

Primary Partition Table

Backup Partition Table

>2.2 Tb support, Unique GUID signature support in partition table HDR

See Section 5 of the UEFI 2.3.1 Spec.

Plumbers conference 2012
GPT Advantages over MBR Partition Table

- **64-bit LBA**
  - No more 2.2TB limit
  - Up to 9.8 zettabytes

- **Improved partitioning**
  - Supports unlimited number of partitions
  - Uses a primary and backup table for redundancy
  - Defines a GUID for identifying each partition
  - Uses GUID & attributes to define partition type

- Uses version number and size fields for future expansion.

- Uses CRC32 fields for improved data integrity

- Each partition contains a 36 Unicode character human readable name.

- No MBR problems
  - No “magic code” must execute as part of booting
UEFI Terminology

UEFI Specification - Key Concepts

- Objects - manage system state, including I/O devices, memory, and events
- UEFI System Table - data structure with data in-formation tables to interface with the systems
- Handle Database and Protocols - callable interfaces that are registered
- UEFI Images - the executable content format
- Events - the software can be signaled in response to some other activity
- Device Paths - a data structure that describes the hardware location of an entity
**GUID**

- Globally Unique Identifier
  - 128-bit quantity defined by Wired for Management (WfM) 2.0 specification

- Used to identify protocols
  - 1:1 with interfaces

- Regulate extension mechanism
  - Documented in the spec
  - Added through drivers

**Safe co-existence of 3rd party extensions**

**http://www.intel.com/design/archives/wfm/index.htm**
Handles

- All protocols have an associated handle
- Every device and executable image in UEFI has a handle protocol in the handle database
- Every boot device must have a device path protocol to describe it
Protocols (API)

- GUID, Interface Structure, Services
  - DEVICE_PATH, DEVICE_IO, BLOCK_IO, DISK_IO, FILE_SYSTEM, SIMPLE_INPUT, SIMPLE_TEXT_OUTPUT, SERIAL_IO, PXE_BC, SIMPLE_NETWORK, LOAD_FILE, UNICODE_COLLATION

UEFI Terminology
Device Path Protocol

- A data structure description of where a device is in the platform
- All boot devices, logical devices and images must be described by a UEFI device path
- The UEFI Specification defines six types of device paths
Six Types of Device Path Types

- Hardware – where is the device in the system
- ACPI – UID/HID of device in AML
- Messaging – Classifies device as LAN, Fiber Channel, ATAPI, SCSI, USB, ...
- Media – i.e. Hard Drive, Floppy or CD-ROM
- BIOS Boot Specification – used to point to boot legacy operating systems
- End of hardware – marks end of device path
Device Path Examples

Acpi(PNP0A03,0\(^1\))/Pci(1F|1)/Ata(Primary,Master)/HD(Part3, Sig00110011\(^2\))

Acpi(PNP0A03,1)/Pci(1E/0)/Pci(0|0)/Mac(0002B3647D69)

Acpi(PNP0A03,0)/Pci(1F|0)/Acpi(PNP0501,0)/UART(115200 81)

See § 9 UEFI 2.X Spec.

\(^1\) ACPI Name space - contain HID, CID, and UID fields that match the HID, CID, and UID values that are present in the platform’s ACPI tables

\(^2\) Truncated to fit on slide, GUIDs are 128 bits
Why UEFI Device Path?

- The UEFI Device Path describes a boot target
  - Binary description of the physical location of a specific target

```
Acpi(PNP0A03,0) /Pci(1F|1) /Ata(Primary,Master) /HD(Part3, Sig010...) /EFI\Boot"/"OSLoader.efi"
```

---

**UEFI Terminology**

**Note: Boot Sequence**
Bootstrap Services

- Full set of firmware services for pre-boot
- Events and notifications
  - Polled devices, no interrupts
- Watchdog timer
  - Elegant recovery
- Memory allocation
- Handle location – for finding protocols
- Image loading
  - For drivers, applications & the OS loader
Runtime Services

- **Minimal set of services for the UEFI Aware OS**
- **Available in boot services and at OS runtime**
- **Timer, Wakeup Alarm**
  - Allows system to wake up or power on at a set time
- **Variables**
  - Boot manager handshake
- **System reset**
ExitBootServices()

Before ExitBootServices() Call

Both boot services & runtime services are available

Exit Boot Services() Call

Issued by the UEFI OS Loader

After ExitBootServices() Call

Only runtime services are available
UEFI Driver Design

Typical System

Processor

PCI Host Bus

PCI Bus

USB

USB Bus

IDE

IDE Bus

VGA

Hard Drive

CD-ROM

PCI-PCMCIA

Floppy Drive

PCI-ISA Bridge

ISA Bus

ISA FDC

Bus Controller

Device Controller

Other

See § 2.5 UEFI 2.X Spec.

Plumbers conference 2012
Driver Initialization

- UEFI Driver Handoff State
- Not Allowed to Touch Hardware Resources
- Installs Driver Binding on Driver Image Handle

Created by LoadImage()
Installed in Driver Initialization
Implemented by Driver Writer

Registers Driver for Later Use
Driver Binding Protocol

Driver Image Handle

LOADED_IMAGE

DRIVER_BINDING

Supported()
Start()
Stop()
Version
Example: UEFI ATAPI Driver Stack

UEFI Driver Design

HD Handle

Device Path Protocol

ATAPI Device Path
ACPI(pnp0604,0)/PCI(0,1)/ATA(primary, master)

Image Handle

Device Path Protocol

File System Protocol (FAT)

Disk IO Protocol

Block IO Protocol

UEFI ATAPI Driver

UEFI ATAPI disk drive

IDE ATAPI disk drive
UEFI 2.1 Features

- Added protocols
  - HII (several protocols)
  - Absolute pointer protocol
- New member functions or equivalent
  - Driver Supported Version (for option roms)
  - Extended Simple Text In (more function keys supported)
  - Authenticated Variables
  - Extended SCSI Pass through
  - Signal on configuration change
  - EHCI exclusive ownership
  - Firmware storage device path
  - Hot key registration support
  - Run-time services with interrupts enabled
- Clean-up: e.g. error returns, * vs ** in declarations in several protocols introduced as a result of implementation
UEFI 2.2 Significant Features

- Networking – IPv6
  - IPv6 stack corresponding to existing IPv4 stack
  - Replacement for PXE protocols which are IPv6 compliant and large network friendly
    - Now being worked through IETF
  - Support for more LAN protocols: EAP and VLAN

- Security – Driver signing
  - Added optional ability to create firmware / OS trust relationships
    - Via key exchange
  - More signature combinations
  - Good / Bad list support
  - Platform owner control of denial response
  - Pre-Boot Authentication (PBA) Framework
    - Passwords, Smart cards, Fingerprint sensors, etc.
UEFI 2.2 Other Features

- HII
  - Additional operators for mapping to other standards
  - Page by page security control
  - Animation updates

- EFI_ATA_PASS_THRU Protocol
  - Gives direct access to ATA devices

- UEFI Driver Health
  - Allow for a driver to fix/re-configure (e.g. rebuild RAID set)

- ABI Updates/Clarifications
  - Floating Point/MMX/XMM
  - 16-Byte stack alignment

- EFI_LOAD_FILE2 Protocol
  - Loads non-boot-option EXEs (PCI option ROMs & apps)
  - Modifies LoadImage() behavior

- EFI_LOADED_IMAGE Protocol
  - Associates entire device path with EXE image

- Bug fixes in spec for rest of document
UEFI 2.3 Features

- Two possible views
  - Special release for ARM binding
  - Fairly quick release for items including ARM binding

- Also includes
  - Boot Services protocol for firmware update
    - Mainly for Option ROMs
  - Bug fixes

- Other items on deck
  - Ubiquitous Firmware Update
UEFI Forum Updates

• UEFI Specification
  – Version 2.3.1, Errata A published on Sept. 7, 2011
  – Clarifications from version 2.3.1
  – Additional ECRs are work in progress

• UEFI Self Compliance Tests (SCT)
  – Published a UEFI Winter 2012 Plugfest Release in Feb, 2012
    ▪ Version 2.3.1 compliance test preview
    ▪ Investigating coverage for 2.3.1 Errata A

• Be Ready for Windows* 8
  – UEFI 2.3.1 support
  – UEFI drivers and applications
  – Secure boot (sign the executables)
  – Seamless boot, hybrid boot, fast boot
  – IPv6 and IPv4 network stack
  – UEFI Spring 2012 Plugfest in Taipei (May 8-10), Redmond (July 16-20th)

• PI Specification
  – Version 1.2 Errata C published in October 2011

2012 marks the ubiquitous adoption of UEFI on PCs
Intel® UDK2010 SR1 (UEFI 2.3.1)

- User Identity (UID) Support (UEFI 2.3.1a)
- Secure Storage Protocol
  - Enable Opal/eDrive SATA devices using the EFI_STORAGE_SECURITY_COMMAND_PROTOCOL, ATA-8 Trusted Send/Receive and IEEE1667 Silo (UEFI 2.3.1a)
- Networking Improvements
  - Errata related to Netboot6-DUID
  - Provide more DHCP4 & DHCP6 API support
  - iSCSI (ip6) open source implementation for IPv6
- Support ATA Asynchronous Block io (UEFI 2.3.1a)
- USB 3.0 Controller Support (XHCI)
- Update Internal Forms Representation (IFR) implementation to match UEFI 2.3.1 Specification
- Fast boot support (asynchronous blockIO2)

www.intel.com/UDK
Class 0

Legacy BIOS

UEFI CSM* only

Class 2

Legacy
BIOS

UEFI Switch: CSM & UEFI

Class 3

UEFI only

* - CSM: Compatibility Support Module or Legacy BIOS written as an UEFI driver

Class 2 configurations (default BIOS setup)
A - CSM and UEFI 2.0-2.1 posted during boot
B - UEFI 2.0-2.1 only posted during boot, CSM disabled
C - UEFI 2.3.1 only posted during boot, CSM disabled, secure boot enabled – only signed modules executed

A, B Shipping today (2012)
C Required by Windows 8 logo
Windows 8 Boot Flow

- Windows 8 installs UEFI OS Loader if UEFI is detected
- Most PCs today boot through CSM path
- For compatibility the CSM boot path available
- Windows 8 logo requirement to boot UEFI only (cannot run csm in client builds)
- Client must boot with UEFI secure boot enabled
- For server if implemented
Windows 8 aims to support <10s boot, on SSD systems
- POST: <2s (without TPM; SSD)
- Resume: <4s (without CSM)
- Device Init: <2s (varies by quality of driver)

New WHQL Requirements for hardware design
- TPM: <300ms init
- Total time 2.3 seconds max for boot with SSD and UEFI secure boot
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 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

UEFI PI will measure OS loader & UEFI drivers into iTMP PCR (Platform Configuration Register)
UEFI/PI Architecture Boot Flow – Create/Evaluate Integrity List

TPM Init
SHA1 Algo
Physical Presence
Measure FV_MAIN
PCR, Event Log

OpROM Scan
Measure ROMs
Update PCR2, Log Event.

Legacy Boot, Measure IPL, Update PCR4, Log Events etc.

ASL Code

Measure & Create
Log in TPM
nvram

SEC
PEI
DXE
BDS
OS

Measure Into PCR’s

Plumbers conference 2012
Measured items in UEFI in Trusted Boot

- PCR4
- PCR0
- PCR1
- PCR2
- PCR8

PC Rx = Register in TPM

Standardized way to measure and report
BitLocker™ Drive Encryption
Static Root of Trust Measurement of early boot components

UEFI Firmware
- TPM Init
- EFI Core
- EFI Extensions

Static PreOS
- Boot manager

All Boot Blobs Unlocked
- OS loader

Volume Blob of Target OS unlocked
- Start OS

UEFI Windows* is using SRTM

Plumbers conference 2012
UEFI 2.3.1 Secure Boot
Intel® UDK2010 SR1 Security Features

- **UEFI Secure Boot**
  - UEFI variable support for UEFI Secure Boot as defined by UEFI 2.3.1a (EFI_VARIABLE_TIME_BASED_AUTHENTICATED_WRITE_ACCESS attribute with EFI_VARIABLE_AUTHENTICATION_2 and EFI_VARIABLE_AUTHENTICATION support)
  - DXE Image Verification library to support UEFI Secure Boot (UEFI 2.3.1a)
  - PK x509 Certificate Support
  - Support EFI_VARIABLE_AUTHENTICATION_2 for PK variable format (UEFI 2.3.1a)
  - Add enable/disable mechanism for UEFI Secure Boot

- **TCG Trusted Boot**
  - TCG EFI Platform Specification
Secure Boot – Three Components

1. Authenticated Variables

2. Driver Signing

3. System-Defined Variables
UEFI Authenticated Variables

- Uses standard UEFI Variable Functions
- Available Pre-boot and also Runtime
- Typically stored in Flash
- Variable Creator signs Variable Hash with Private Key (PKCS-7 Format)
- Signature & Variable Passed Together for Create, Replace, Extend, or Delete
- Several System-defined variables for Secure Boot

Extensible Integrity Architecture
Updating Authenticated Variable

- Support for Append added (UEFI 2.3.1)
- Counter-based authenticated variable (UEFI 2.3)
  - Uses monotonic count to against suspicious replay attack
  - Hashing algorithm - SHA256
  - Signature algorithm - RSA-2048
- Time-based authenticated variable (UEFI 2.3.1) *
  - Uses timestamp as rollback protection mechanism
  - Hashing algorithm - SHA256
  - Signature algorithm - X.509 certificate chains
    - Complete X.509 certificate chain
    - Intermediate certificate support (non-root certificate as trusted certificate.

* only Time-based authenticated variables implemented in Tianocore.org
UDK2010 SR1
Secure Boot – Three Components

1. Authenticated Variables
2. Driver Signing
3. System-Defined Variables
UEFI Driver Signing

- UEFI Driver Signing Utilizes Microsoft* Authenticode* Technology to sign UEFI executables
- In Secure Boot, signatures should be checked:
  1. UEFI Drivers loaded from PCI-Express Cards
  2. Drivers loaded from mass storage
  3. Pre-boot EFI Shell Applications, f/w updaters
  4. OS UEFI Boot-loaders
- UEFI Signing is not applied to
  1. Drivers in the Factory BIOS
  2. Legacy BIOS components (also known as CSM)
  3. CSM must be disabled in boot for system to be secure (UEFI boot)
    - CSM can be enabled in setup for non-UEFI boot options
    - Shell is not considered a secure boot option

Enhanced by UEFI 2.3.1
Secure Boot – Three Components

1. Authenticated Variables

2. Driver Signing

3. System Defined Variables
## Secure Boot Authenticated Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PK</td>
<td><strong>Platform Key</strong> – Root key set to enable Secure Boot</td>
</tr>
</tbody>
</table>
| KEK      | **Key Exchange Key**  
|          | List of Cert. Owners with db, dbx update privilege |
| db       | List of Allowed Driver or App. Signers (or hashes) |
| dbx      | List of Revoked Signers (or hashes) |
| SetupMode| 1 = in Setup Mode, 0 = PK is Set (User Mode) |
| SecureBoot| 1 = Secure Boot in force |

**Notes:**
- Owner of cert. in KEK can update db, dbx
- Owner of cert. in PK can update KEK

*UEFI Defines System Databases for Secure Boot*
Secure Boot – Three Components

1. Authenticated Variables
2. Driver Signing
3. System Defined Variables

\[ \text{UEFI 2.3.1 SECURE BOOT} \]
Secure Boot Begins @ the Factory

Pre-production | Production | User

Certificate Generating Station @ OEM

1. OEM collects certificates provided by OSVs, Partners, and OEM’s own keys.

2. "DB Generator" creates the Initial Security Load for new computers.

3. Initial Security Load is installed onto each computer at the factory, enabling Secure Boot.
   1) Initial db and dbx
   2) KEK with allowed updaters
   3) Platform Key (PK)

After delivery, the OEM or OSV can update with new certificates or revoked certificates (dbx)

OEM Responsible for Initializing Secure Boot and can allow user to disable Secure Boot or add KEK, PK, DB in setup

Plumbers conference 2012
Secure Boot Protects the User

User attempts to boot a compromised system → OS Boot-loader image checked against pre-loaded database → Root-kit fails checks, user protected by Secure Boot

Secure Boot Tests Signatures to Reject Potential Threats
For more information - UEFI Secure Boot


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

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
Tunnel Mountain Intel DQ57TM UEFI 2.3.1 platform

Intel® UDK 2010 SR1 UP1 Compatible, supports UEFI 2.3.1 (updates match tianocore.org open source)

Pre-assembled systems available at HDNW, visit

http://www.Tunnelmountain.net based on DQ57TM

tomk@hdnw.com, (425) 943-5515 ext 42234. Use product name “Tunnel Mountain” when ordering

- Comes with class 2 CSM and UEFI enabled firmware
- Download site has UEFI class2 firmware (csm not on by default)
- Comes with serial port for debug
- Can be ordered with optional ITP connector and socketed SPI flash
  * Romley Server UEFI 2.3.1 to be available Sep ’12
  * Maho Bay DQ77MK Q4 ’12
- * - not available yet in validation test

Visit http://www.UEFIDK.com for the latest collateral on UEFI systems

For UEFI developers
Getting your own UEFI 2.3.1 systems

Intel Production motherboards
UEFI 2.3.1 enabled Windows 8 client desktop motherboards
Sept - Oct 2012 rollout for 7 series Ivybridge systems (DH77, DQ77 etc.)
Goto Intel.com under support to update BIOS on motherboards

Ivybridge Ultrabooks certified for Windows 8

Contact your favorite BIOS vendor and ask for a UEFI 2.3.1 enabled board
What Vintage of UEFI is your system?

Get into BIOS setup
IS there a switch for CSM or legacy bios (to turn it off or force UEFI only)?
Goto UEFI shell type “ver” -> UEFI system spec revision (2.3.1?)

Does the system contain drivers for boot/console devices in system?
UEFI shell -> drivers command

Get SCT’s from UEFI.org and run them (Do they pass or which tests fail?)
is it one the OS needs?

Goto Ubuntu’s Linux Firmware Test suite - Run the UEFI suite to see if it passes. Contribute effort to test UEFI!
Git://kernel.ubuntu.com/hwe/fwts
UEFI Industry Resources

UEFI Forum
www.uefi.org

UEFI Open Source
www.tianocore.sourceforge.net

Intel UEFI Resources
www.intel.com/UDK

Intel EBC Compiler

UEFI Books/ Collateral
www.intel.com/intelpres:

Training/IHVs Contact
Laurie Jarlstrom
Intel UEFI Training
Laurie.Jarlstrom@intel.com

Brian Richardson
Intel IHVs UEFI Support
Brian.Richardson@intel.com

Plumbers conference 2012
Backup
**UEFI Image (driver & application/OS loader) Signing**

- **Why?** - Origin & Integrity
- **How?** - AuthentICODE PE/COFF

---

**PE Image**

- **PE Header**
  - Certificate Directory
- **Section 1**
  - ......  
  - Section N
- **Type**
  - Attribute Certificate Table

---

**PKCS#7 + AuthentICODE Ext**

- **ContentInfo**
  - PE file hash
- **Certificate**
  - X.509 Certificate
- **SignInfo**
  - Signed hash of ContentInfo
UEFI Authenticated Variable

- **Why?** - Integrity (no confidentiality)
- **How?** - Time Based

Authenticated Variable

- **Input Variable Data Authentication**
  - **Time Stamp**
  - **Type**
  - **Certificate**

- **Data Content**

PKCS#7 ContentInfo
- N/A

Certificate
- X.509 Certificate

SignInfo
- Signed hash of VariableName + VariableGuid + Attributes + TimeStamp + DataContent
### Secure Boot’s Authenticated Variables

<table>
<thead>
<tr>
<th>Key/ DB Name</th>
<th>Variable</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>PkPub</td>
<td>PK</td>
<td>OEM and Platform FW - format is RSA-2048</td>
</tr>
<tr>
<td>Key Exchange Key</td>
<td>KEK</td>
<td>Platform FW and OS - format is RSA-2048</td>
</tr>
<tr>
<td>Authorized Signature DB</td>
<td>DB</td>
<td>Authorized Signing certificates - white list</td>
</tr>
<tr>
<td>Forbidden Signature DB</td>
<td>DBX</td>
<td>Unauthorized Signing certificates - Black list</td>
</tr>
</tbody>
</table>
| 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  |

- Do NOT generate your own private keys
- Use FIPS certified key generators (hardware crypto)
- Protect your private keys (under physical protection)
**Authorization Flow**

1. **Enroll**
   - **Authenticated Variable**
     - PK
     - KEK
     - db
     - dbx
   - **Variable**
   - **DXE FV**
     - Image Verify

2A. **Signed Image Discover**
   - OpRom.efi
   - Certificate + SignInfo

2B. **Signature Verification**
   - OsLoader.efi
   - Certificate + SignInfo

2C. Signed Image Load And measure Into TPM
Put them altogether: UEFI Secure Boot

Custom Mode
Edit keys on x86
End user controls - Custom Secure Boot Options

- Enrolling DB and/or DBX for physically present user
# Disable Secure Boot

1. Select Custom Secure Boot Options
2. Select PK Options
3. Delete Pk (space bar)
4. Reset

<table>
<thead>
<tr>
<th>Security Options</th>
<th>PK Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attempt Secure Boot</td>
<td>PK Options</td>
</tr>
<tr>
<td>Secure Boot Mode</td>
<td>Enroll PK</td>
</tr>
<tr>
<td>Custom Secure Boot Options</td>
<td>Delete Pk</td>
</tr>
</tbody>
</table>

Choose to Delete PK, Otherwise keep the PK

Configuration changed. Reset to apply it Now. Press ENTER to reset.
UEFI Secure Boot Database Review

Variables are write protected and cannot be erased from std UEFI variables services or out of band.

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

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

Update Enable

PK

Update Enable

KEK

Update Enable

db

Update Enable

dbx
Public vs. Private Keys

- A pair of keys, one public, one private, are created
- Private keys stay secure at Partner or in the OEM’s Security Office
- Private keys are used to ‘sign’ objects
- Only Public keys loaded into the Platform
- Public keys are used to check signatures

Private Keys Must be Stored Securely!
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, Msft
  Optional: Include 2\textsuperscript{nd} key created by OEM

- **db** – OS Partner supplies Key, win8 msft
  CA Partner supplies Key, UEFI CA hosted by msft
  Optional: OEM App Signing Key
  Bios update key should not show up in DB, kek or PK

**Signature Tests using db Keys Block Rogue S/W!**
OEM Administration

- Keys are installed for testing with target OS
- Keys are installed in the factory before shipping

- Preparation Tasks
  1. Gather public keys from partners
  2. Generate PK for model
  3. Make a package of initial key load
  4. Occasional maintenance of forbidden list

- Repetitive Tasks
  1. Factory will boot and install the initial key load

Careful Preparation Delivers Successful Launch