
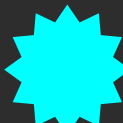



# UEFI & EDK II Training

UEFI Aware Operating System

[tianocore.org](https://tianocore.org)

# LESSON OBJECTIVE

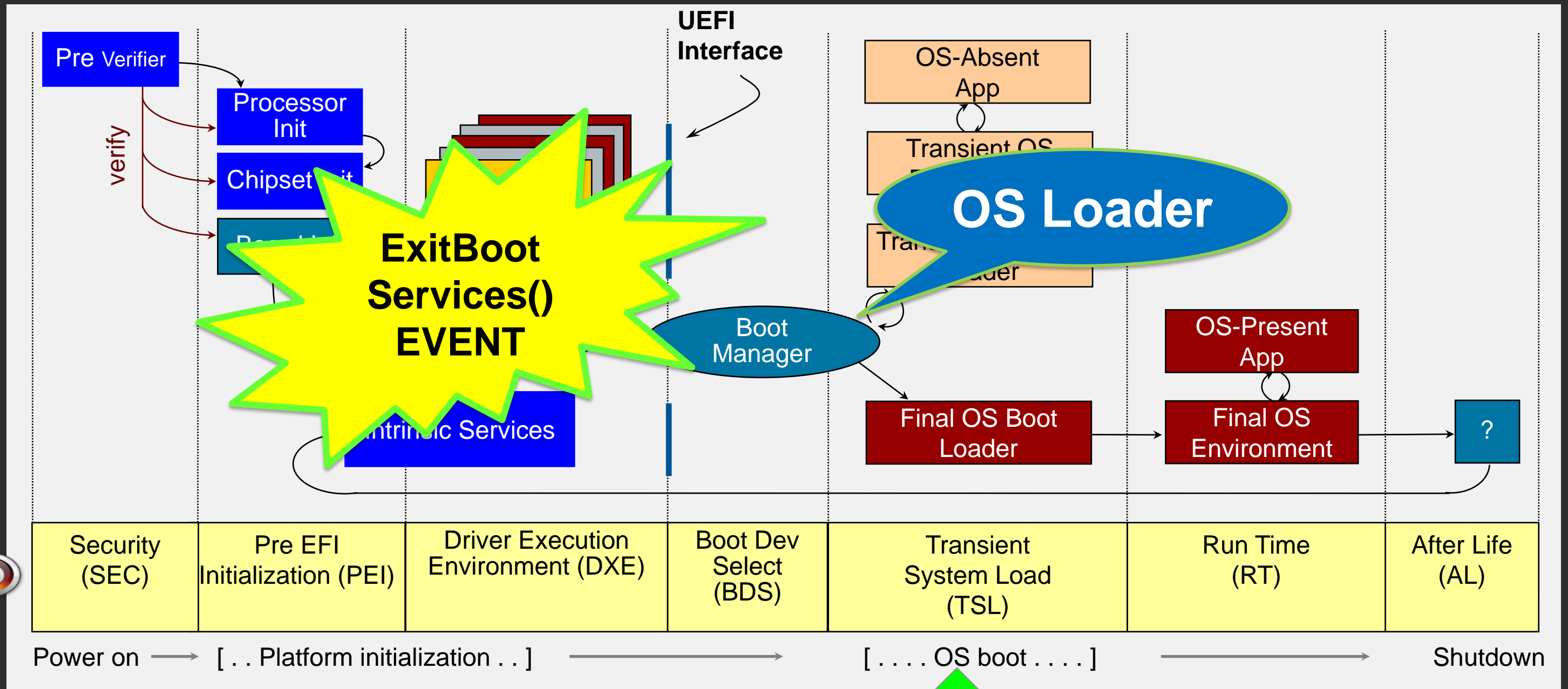
-  Explain How the OS and UEFI Work together
-  Explain the UEFI Requirements for UEFI aware OS
-  Explain How Secure Boot Fits with UEFI

# UEFI AWARE OS REQUIREMENTS

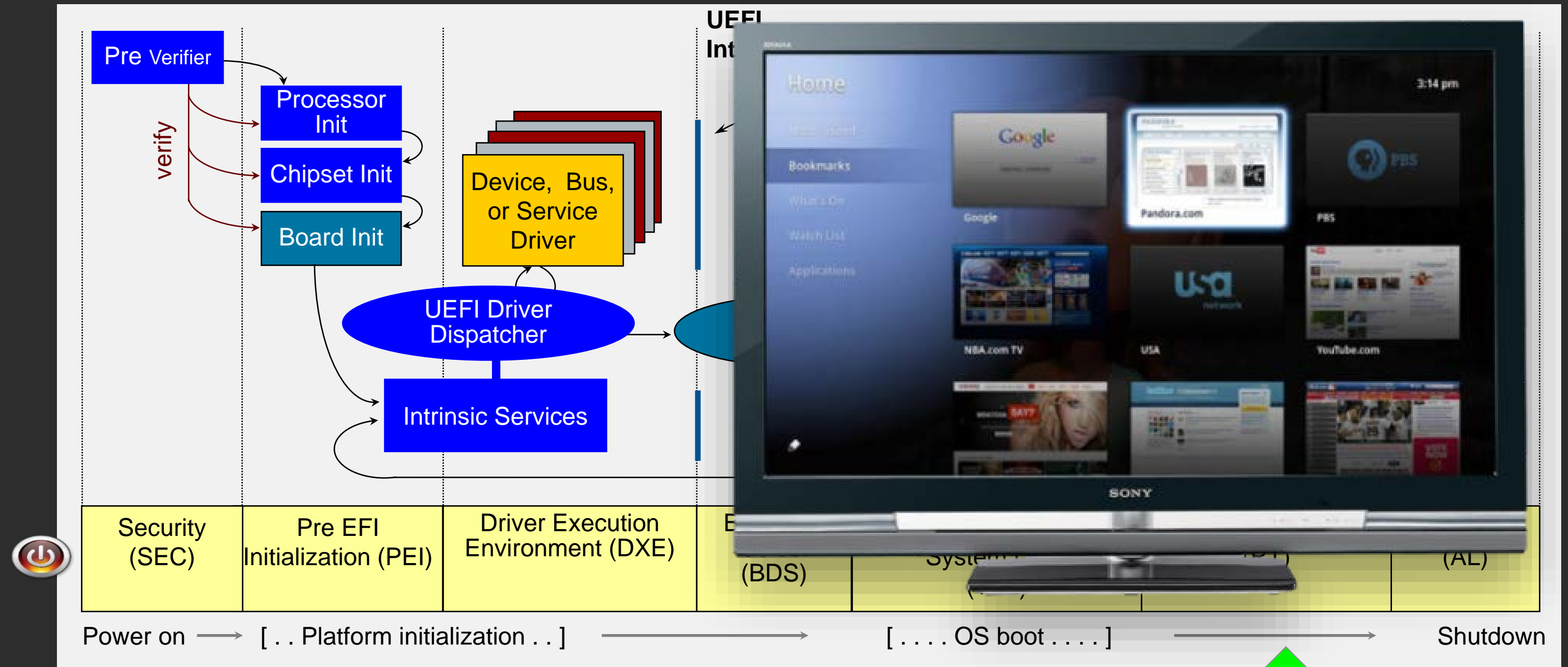
## Common Requirements

# UEFI OPERATING SYSTEMS









# UEFI OS REQUIREMENTS

UEFI Drivers:  
Boot devices/console

UEFI OS installer

UEFI OS Loader

Disk  
Partition/Formats

Firmware  
Requirements

Set Boot Path to  
Boot to UEFI OS

# UEFI System Classes (based on firmware interfaces)

## UEFI Class 0

- Boots Legacy - int 19 ONLY
- Legacy BIOS Only (16 bit)
- No UEFI or UEFI PI Interfaces

## UEFI Class 1

- Boots Legacy - int 19 ONLY
- Uses UEFI / PI Interfaces
- Only legacy BIOS runtime Interfaces

## UEFI Class 2

- Boots Legacy - int 19 or UEFI
- Uses UEFI / PI Interfaces
- Legacy BIOS runtime Interfaces w/ **CSM**

## Limited Benefits

- ✓ OEMs / ODMs Internal
- ✓ Double code development
- ✓ Compromised security – MBR exposure



# UEFI System Classes (based on firmware interfaces)

## UEFI Class 0

- Boots Legacy - int 19 ONLY
- Legacy BIOS Only (16 bit)
- No UEFI or UEFI PI Interfaces

## UEFI Class 1

- Boots Legacy - int 19 ONLY
- Uses UEFI / PI Interfaces
- Only legacy BIOS runtime Interfaces

## UEFI Class 2

- Boots Legacy - int 19 or UEFI
- Uses UEFI / PI Interfaces
- Legacy BIOS runtime Interfaces w/ **CSM**

## UEFI Class 3

- Boots **ONLY** UEFI
- Uses UEFI / PI Interfaces
- Runtime exposes only UEFI interfaces

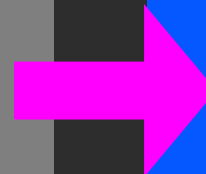
# UEFI System Classes (based on firmware interfaces)

## Full Benefits

- ✓ UEFI Innovation
- ✓ Smaller code size/  
Validation
- ✓ Extensibility

## Only Class after 2020

Enabling *Secure Boot*  
creates another Class



## UEFI Class 3 +

- Boots **ONLY** UEFI
- Uses UEFI / PI Interfaces
- Runtime exposes only UEFI interfaces

UEFI Secure Boot “ON”

# Required UEFI Drivers: OS Install & Boot

**Boot Device**

**Console Output**

**Console Input**

**NVRAM Driver**

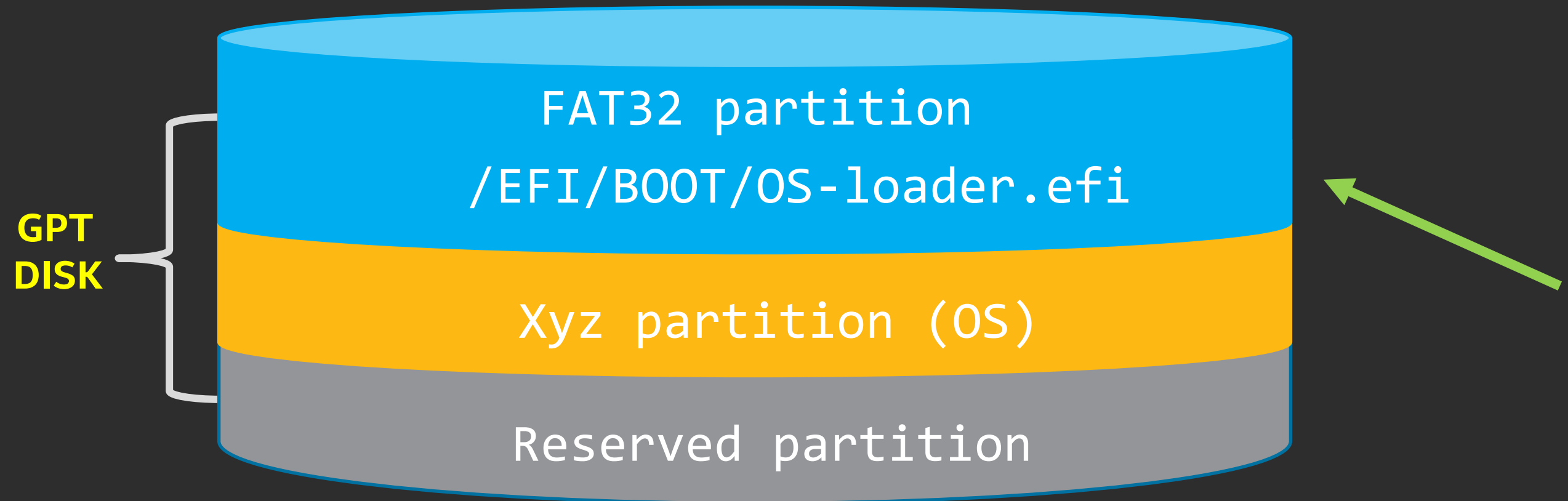
# UEFI OS LOADER

- OS install process includes UEFI loader
  - `/efi/boot/bootx64.efi` `/efi/redhat/grub.efi`
- Call UEFI boot & runtime services to start OS
- Exit UEFI Boot Services
- Transfer control to native OS

# UEFI OS INSTALLER

- Discover UEFI storage devices
- Setup storage device: GPT w/ FAT32 boot partition
- Create boot variables `BootXXXX` and set the `BootNext`

# Disk Partition and Format

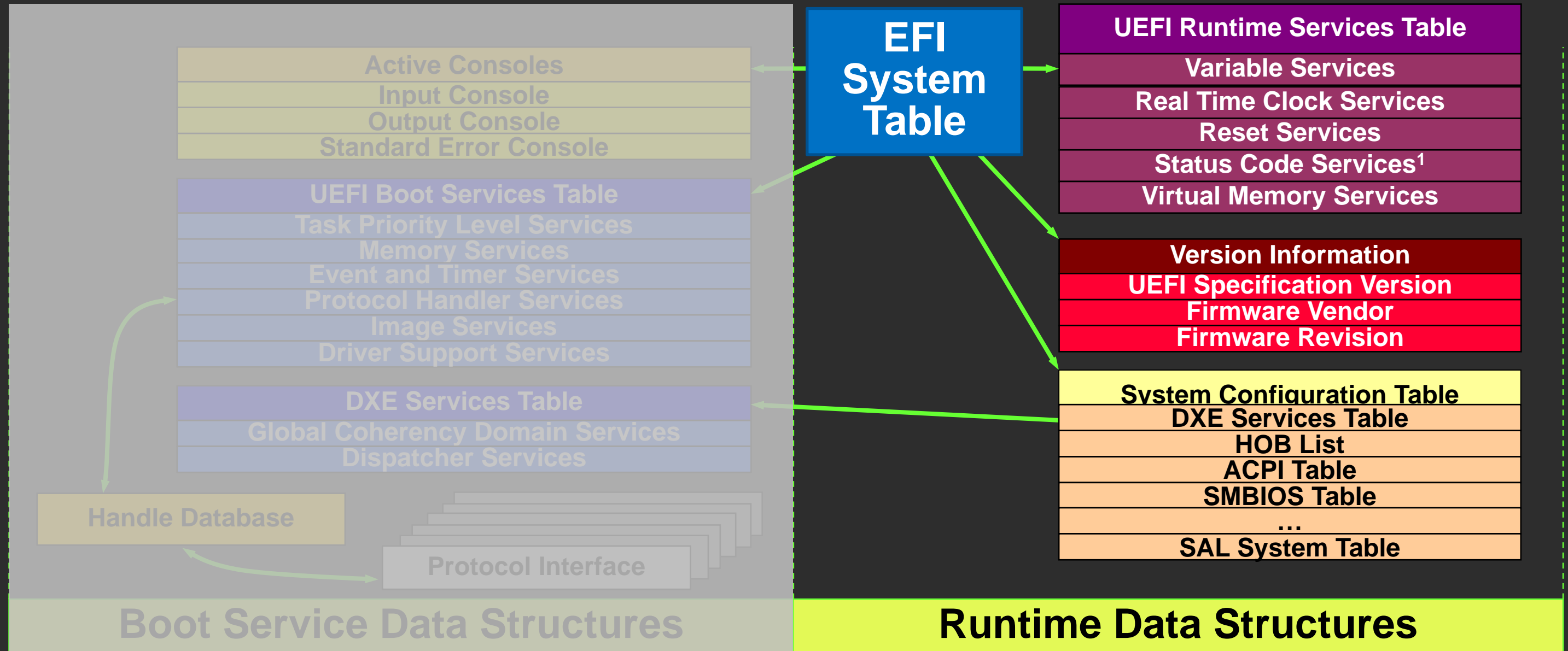


# INTERFACE INSIDE OS RUNTIME

UEFI Runtime Services



# Runtime Services Available to the UEFI Aware OS





# Accessing RT services from Windows API

- GetFirmwareEnvironmentVariable: [MSDN Link](#)
- SetFirmwareEnvironmentVariable: [MSDN Link](#)
- Example: (determine if UEFI or Legacy BIOS)

```
int main(int argc, char*argv[])
{
    GetFirmwareEnvironmentVariableA("",
        "{00000000-0000-0000-0000-000000000000}", NULL, 0);
    if (GetLastError() == ERROR_INVALID_FUNCTION) {
        printf("Legacy"); // This.. is.. LEGACY BIOS....
        return 1;
    } else{
        printf("UEFI"); // This.. is.. UEFI
        return 0;
    }
    return 0;
}
```



# Accessing RT services from Linux OS

Firmware Test Suite, it includes a Linux kernel driver to help with it's interactions with UEFI. Note that this is a Linux-centric test suite, solution won't work for other OSes.

- <http://kernel.ubuntu.com/git/hwe/fwts.git>
- <https://bugs.launchpad.net/ubuntu/+source/linux/+bug/1633506>
- <https://patchwork.kernel.org/patch/9323781/>
- <http://www.basicinputoutput.com/2016/03/introduction-to-firmware-test-suite-fwts.html>

# SECURITY WITH UEFI

How does UEFI ensure the Operating System is trusted?

Security Resources: <https://github.com/tianocore/tianocore.github.io/wiki/EDK-II-Security-White-Papers>

# BOOT SECURITY TECHNOLOGIES

**Hardware Root of Trust**

Boot Guard, Intel® TXT

**Measured Boot**

Using TPM<sup>1</sup> to store hash values

**Verified Boot**



Boot Guard +  
UEFI Secure Boot

<sup>1</sup>TPM – Trusted Platform Module

Resources: <https://firmwaresecurity.com/2015/07/29/survey-of-boot-security-technologies/>

# HARDWARE ROOT OF TRUST

## Boot Guard

CPU verifies signature  
Verification occurs before system FW starts  
Hash of public key is fused in CPU

Verification

## Intel® TXT

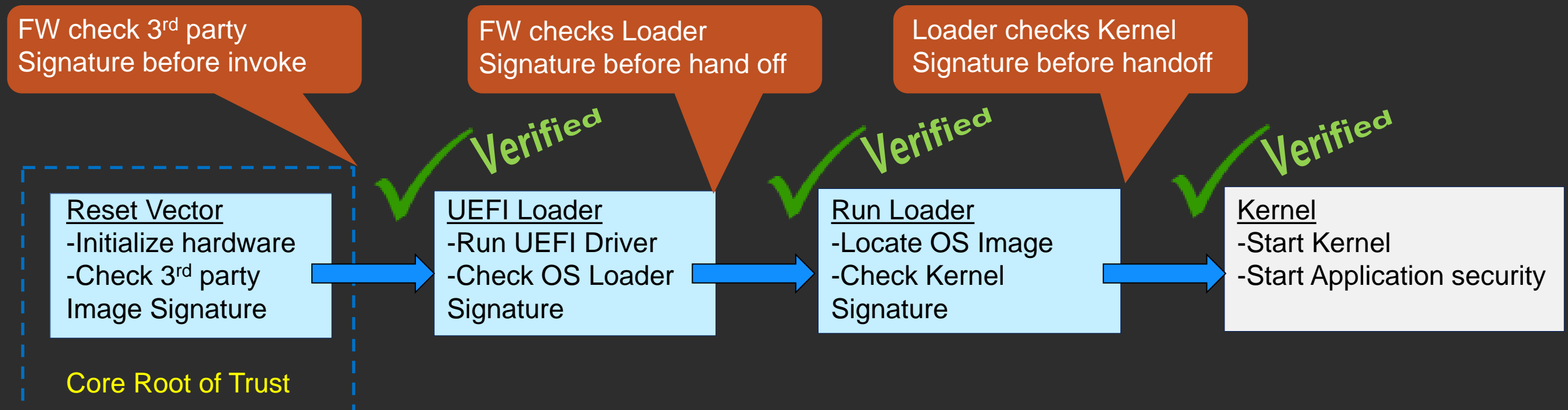
Uses a Trusted Platform Module (TPM) & cryptographic  
Provides Measurements

Measurements



## Software ID checking during every step of the boot flow:

1. UEFI System FW (updated via secure process)
2. Add-In Cards (signed UEFI Option ROMs)
3. OS Boot Loader (checks for “secure mode” at boot)



# AUTHENTICATED VARIABLES

PK

KEK

DB

DBX

SetupMode

SecureBoot

```
2.0 Shell> dmpstore SecureBoot
```

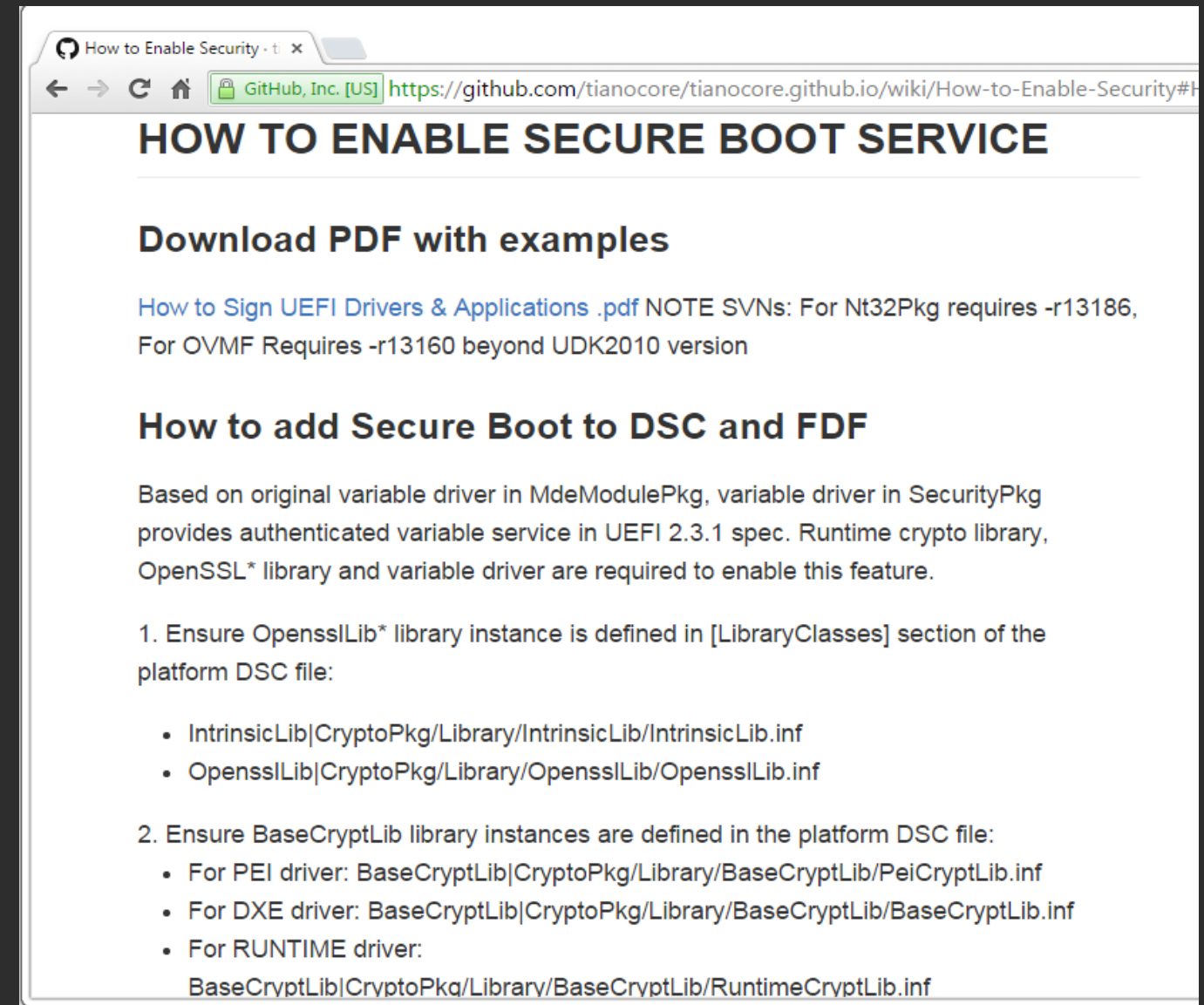
```
Variable - RS+BS - '8BE4DF61-93CA-11D2-AA0D-00E098032B8C:SecureBoot' - DataSize  
= 0x01
```

```
00:
```

```
00 * *
```

# Security Package Project Page [Wiki Link](#)

- Wiki Link: [How-to-Enable-Security](#)
- PDF: [How to Sign UEFI Images V1.31](#)
- [Beyond BIOS UEFI Secure Boot](#)
- Build command line switch -  
SECURE\_BOOT\_ENABLE = TRUE
- Install the OpensslLib CryptoPkg :  
From edk2: “git submodule update --init”



The screenshot shows a web browser window with the URL <https://github.com/tianocore/tianocore.github.io/wiki/How-to-Enable-Security#h>. The page title is "HOW TO ENABLE SECURE BOOT SERVICE". Below the title, there is a section "Download PDF with examples" with a link to "How to Sign UEFI Drivers & Applications .pdf". A note states: "NOTE SVNs: For Nt32Pkg requires -r13186, For OVMF Requires -r13160 beyond UDK2010 version". Another section is titled "How to add Secure Boot to DSC and FDF". The text explains that based on the original variable driver in MdeModulePkg, the variable driver in SecurityPkg provides authenticated variable service in UEFI 2.3.1 spec. It mentions that the Runtime crypto library, OpenSSL\* library, and variable driver are required to enable this feature. Two numbered steps are provided: 1. Ensure OpensslLib\* library instance is defined in [LibraryClasses] section of the platform DSC file, with a bulleted list of file paths: IntrinsicLib|CryptoPkg/Library/IntrinsicLib/IntrinsicLib.inf and OpensslLib|CryptoPkg/Library/OpensslLib/OpensslLib.inf. 2. Ensure BaseCryptLib library instances are defined in the platform DSC file, with a bulleted list of file paths: For PEI driver: BaseCryptLib|CryptoPkg/Library/BaseCryptLib/PeiCryptLib.inf, For DXE driver: BaseCryptLib|CryptoPkg/Library/BaseCryptLib/BaseCryptLib.inf, and For RUNTIME driver: BaseCryptLib|CryptoPkg/Library/BaseCryptLib/RuntimeCryptLib.inf.

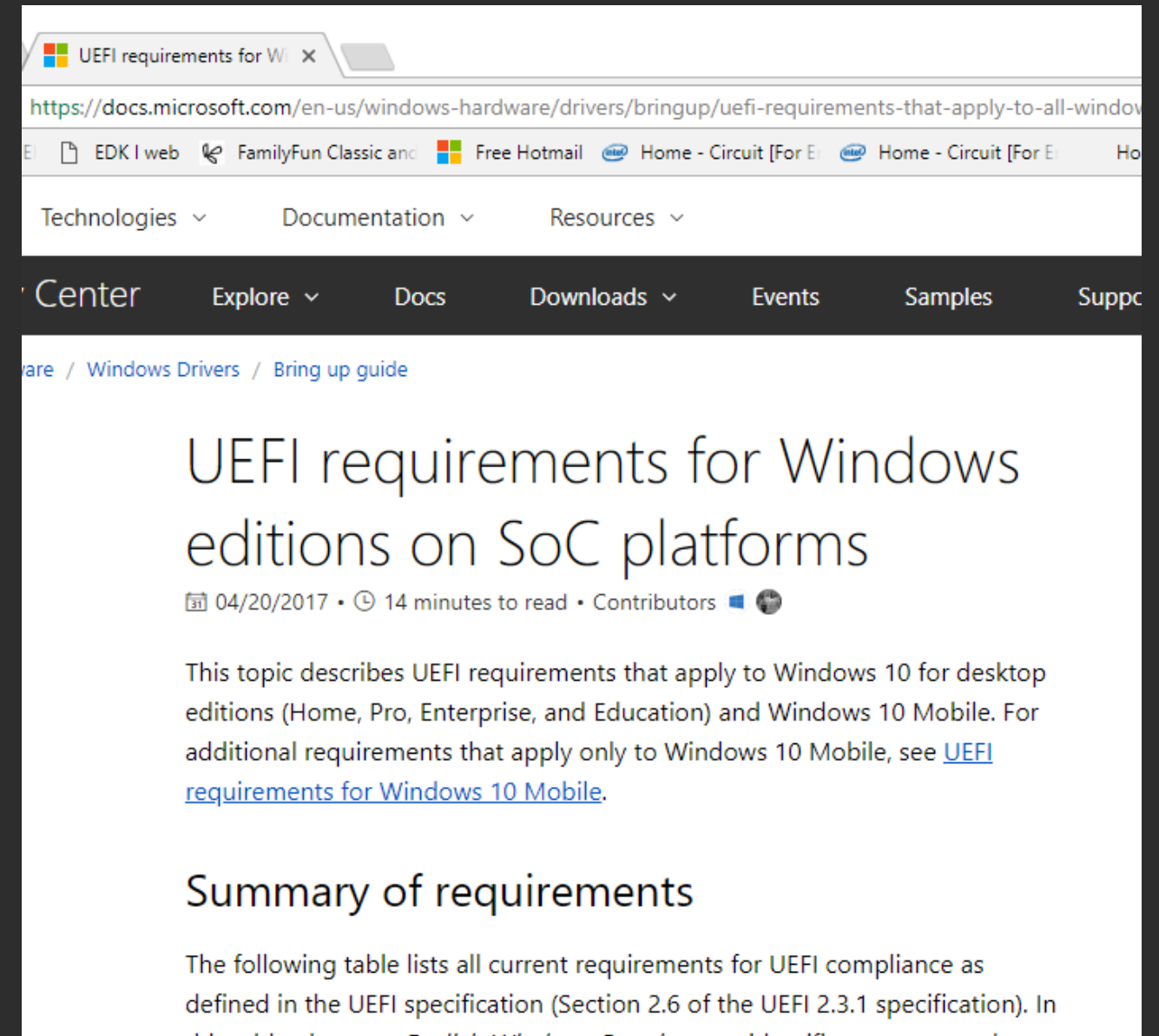
# Windows Secure Boot Key Creation and Management Guidance

- Windows - Secure Boot Key Creation & Management Guide
- Creation and management of the Secure Boot keys and certificates in a manufacturing environment.
- Addresses questions related to creation, storage and retrieval of Platform Keys (PKs), secure firmware update keys, and third-party Key Exchange Keys (KEKs).



# Many Platforms are Requiring UEFI Secure Boot Enabled

- Secure Boot now mandated for specific platforms
- See “Security requirements” on UEFI requirements for Windows editions on SoC Platforms



# SUMMARY

- ★ Explain How the OS and UEFI Work together
- ★ Explain the UEFI Requirements for UEFI aware OS
- ★ Explain How Secure Boot Fits with UEFI



# Questions?



# RETURN TO MAIN TRAINING PAGE



Return to Training Table of contents for next presentation [link](#)





# ACKNOWLEDGEMENTS

Redistribution and use in source (original document form) and 'compiled' forms (converted to PDF, epub, HTML and other formats) with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code (original document form) must retain the above copyright notice, this list of conditions and the following disclaimer as the first lines of this file unmodified.

Redistributions in compiled form (transformed to other DTDs, converted to PDF, epub, HTML and other formats) must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS DOCUMENTATION IS PROVIDED BY TIANOCORE PROJECT "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL TIANOCORE PROJECT BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS DOCUMENTATION, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright (c) 2021, Intel Corporation. All rights reserved.

# BACKUP

- Deficiency: Boot path malware targets
- UEFI and Secure Boot harden the boot process
- Firmware/software in the boot process must be signed by a trusted Certificate Authority (CA)
- Firmware image is hardware-protected
- 3<sup>rd</sup> party drivers signed using CA-holding trusted keys
- Trusted signing key's database factory-initialized and OS-updated



# WHY??? SECURE BOOT WITH UEFI

## 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



# UEFI SECURE BOOT FLOW

