**# Title**: Device Security Protocol

**# Status**: Submitted to industry standard forum

**# Document**: UEFI Platform Initialization Specification Version 1.7

**# License**: SPDX-License-Identifier: CC-BY-4.0

**# Submitter**: [TianoCore Community](<https://www.tianocore.org>)

**# Summary of the change**

**[Background]**

The PI specification defined “**Security Architectural Protocols**” (**EFI\_SECURITY\_ARCH\_PROTOCOL** and **EFI\_SECURITY2\_ARCH\_PROTOCOL**) to verify the executable image in the UEFI environment by the DXE Core.

Today, there is new requirement to not only verify an executable image, but a device on the system. The entity to verify the device might be a standalone platform Root-of-Trust, or the system firmware once the system firmware becomes part of Chain-of-Trust. The Distributed Management Task Force (DMTF) defines Secure Protocol and Data Model (SPDM) specification. The hardware device standard group (such as PCI-SIG and USB) defines the interface to transport the SPDM message for device authentication and measurement. Trusted Computing Group (TCG) also defines the event log for SPDM measurement.

As such, the system firmware needs an architecture way to verify a device on the platform by the device driver in the system firmware, based upon the policy defined by the platform.

**[Proposal]**

This proposal adds a new protocol - **EFI\_DEVICE\_SECURITY\_PROTOCOL**. It is similar to the **EFI\_SECURITY2\_ARCH\_PROTOCOL**. This protocol abstracts the invocation of device firmware authentication and measurement defined by the Distributed Management Task Force (DMTF) Secure Protocol Data Model (SPDM) specification and the Trusted Computing Group (TCG) measured boot.

The platform need produce this protocol as the platform policy on device security. The device driver need consume this protocol to authenticate and/or measure the device before activate and use the device.

Reference:

1. UEFI Specification 2.8 - www.uefi.org
2. PI Specification 1.7 - www.uefi.org
3. DMTF SPDM “Security Protocol and Data Model Specification” - https://www.dmtf.org/standards/pmci
4. USB “USB Authentication Specification” - https://www.usb.org/documents
5. PCI-SIG “Component Measurement and Authentication (CMA) ECR” - https://pcisig.com/specifications/review-zone
6. NIST SP800-193 “Platform Firmware Resiliency Guidelines” - https://csrc.nist.gov/publications/sp800

**# Benefits of the change**

1. We standardize the way to authenticate a device, similar as the way we authenticate a UEFI/PI file. A platform can use a common UEFI/PI security policy driver to verify all device components on the platform, and corresponding action, such as recovery, which meets the NIST SP 800-193 requirement.

**# Impact of the change**

1. Add new device signature database (EFI\_CERT\_X509\_CERT\_CHAIN\_GUID, EFI\_CERT\_X509\_CERT\_CHAIN\_SHA256\_GUID, EFI\_CERT\_X509\_CERT\_CHAIN\_SHA384\_GUID, EFI\_CERT\_X509\_CERT\_CHAIN\_SHA512\_GUID) in UEFI specification Chapter 32 – Secure Boot and Driver Signing. 34.4.1. Signature Database.
2. Extend device signature to existing signature database (EFI\_CERT\_X509\_GUID, EFI\_CERT\_X509\_SHA256\_GUID, EFI\_CERT\_X509\_SHA384\_GUID, EFI\_CERT\_X509\_SHA512\_GUID) in UEFI specification Chapter 32 – Secure Boot and Driver Signing. 34.4.1. Signature Database.
3. Add new device signature variable GUID/Name (devdb, devdbx, devdbt) in UEFI specification Chapter 32 – Secure Boot and Driver Signing. 32.6.2. UEFI Device Signature Variable GUID and Variable Name.
4. No impact to the existing image authentication signature database.
5. No impact to SHA1 related definition. SHA1 is considered as unsecure and not used in device authentication.

# Detailed description of the change [normative updates]

**PI Specification Volume 2.**

13.5 Device Security Protocol  
**EFI\_DEVICE\_SECURITY\_PROTOCOL**

**Summary**

This protocol abstracts the platform policy to authenticate a device. This protocol is optional.

**GUID**

**#define EFI\_DEVICE\_SECURITY\_PROTOCOL\_GUID \**

**{0x5d6b38c8, 0x5510, 0x4458, 0xb4, 0x8d, 0x95, 0x81, 0xcf, 0xa7, 0xb0, 0xd}**

**Protocol Interface Structure**

**typedef struct \_EFI\_DEVICE\_SECURITY\_PROTOCOL {**

**UINT64 Revision;**

**EFI\_DEVICE\_AUTHENTICATION DeviceAuthentication;**

**} EFI\_DEVICE\_SECURITY\_PROTOCOL;**

**Parameters**

*Revision*

The revision of this Device Security Protocol.

*DeviceAuthentication*

This service is called by the device driver to authenticate or and/or measure the device. See the **DeviceAuthentication()** function description

**Description**

The **EFI\_DEVICE\_SECURITY\_PROTOCOL** is used to abstract platform-specific policy from the device driver. This protocol must be produced by a boot service or runtime service driver and may be consumed by the device driver to validate a device before activating or using the device.

**Related Definitions**

**//**

**// Revision The revision to which the DEVICE\_IDENTIFIER interface adhere**

**//          All future revisions must be backwards compatible.**

**//**

**#define EFI\_DEVICE\_IDENTIFIER\_REVISION 0x00010000**

**EFI\_DEVICE\_SECURITY\_PROTOCOL.DeviceAuthentication**

**Summary**

This service is used by a device driver authenticating a device. It allows the system to execute a platform specific policy in response the different authentication state.

**Prototype**

**typedef**

**EFI\_STATUS**

**(EFIAPI \*EFI\_DEVICE\_AUTHENTICATE)(**

**IN EFI\_DEVICE\_SECURITY\_PROTOCOL  \*This,**

**IN EFI\_DEVICE\_IDENTIFIER         \*DeviceId**

**);**

**Parameters**

*This*

The protocol interface pointer.

*DeviceId*

The identifier for the device. See the **DeviceAuthentication()** function description

**Description**

The device driver uses this service to measure and/or verify a device. The flow in device driver is:

  1) Device driver discovers a new device.

  2) Device driver creates an **EFI\_DEVICE\_PATH\_PROTOCOL**.

  3) Device driver creates a device access protocol. e.g. **EFI\_PCI\_IO\_PROTOCOL** for PCI device.  **EFI\_USB\_IO\_PROTOCOL** for USB device.

  4) Device driver installs the **EFI\_DEVICE\_PATH\_PROTOCOL** with **EFI\_DEVICE\_PATH\_PROTOCOL\_GUID**,  and the device access protocol with **EFI\_DEVICE\_IDENTIFIER\_TYPE\_xxx\_GUID**. Once it is done, a DeviceHandle is returned.

  5) Device driver creates **EFI\_DEVICE\_IDENTIFIER** with **EFI\_DEVICE\_IDENTIFIER\_TYPE\_xxx\_GUID** and the DeviceHandle.

  6) Device driver calls DeviceAuthenticate().

  7) If DeviceAuthenticate() returns **EFI\_SECURITY\_VIOLATION**, the device driver uninstalls

     all protocols on this handle.

  8) If DeviceAuthenticate() returns EFI\_SUCCESS, the device driver installs the device access

     protocol with a real protocol GUID. e.g.

**EFI\_PCI\_IO\_PROTOCOL** with **EFI\_PCI\_IO\_PROTOCOL\_GUID**.

**EFI\_USB\_IO\_PROTOCOL** with **EFI\_USB\_IO\_PROTOCOL\_GUID**.

**Related Definitions**

**//**

**// The device identifier.**

**//**

**typedef struct {**

**UINT32                Version;**

**EFI\_GUID              DeviceType;**

**EFI\_HANDLE            DeviceHandle;**

**} EFI\_DEVICE\_IDENTIFIER;**

**Parameters**

*Version*

The version of this data structure.

*DeviceType*

The type of the device.

This field is also served as a device Access protocol GUID.   The device access protocol is installed on the DeviceHandle.   The device access protocol is device specific.   **EFI\_DEVICE\_IDENTIFIER\_TYPE\_PCI\_GUID** means the device access protocol is PciIo.   **EFI\_DEVICE\_IDENTIFIER\_TYPE\_USB\_GUID** means the device access protocol is UsbIo.

*DeviceHandle*

The handle created for this device.

NOTE: This might be a temporary handle.   If the device is not authenticated, this handle shall be uninstalled.   As minimal requirement, there should be 2 protocols installed on the device handle.

1) An **EFI\_DEVICE\_PATH\_PROTOCOL** with **EFI\_DEVICE\_PATH\_PROTOCOL\_GUID**.

2) A device access protocol with **EFI\_DEVICE\_IDENTIFIER\_TYPE\_xxx\_GUID**.

If the device is PCI device, the **EFI\_PCI\_IO\_PROTOCOL** is installed with **EFI\_DEVICE\_IDENTIFIER\_TYPE\_PCI\_GUID**.   If the device is USB device, the **EFI\_USB\_IO\_PROTOCOL** is installed with **EFI\_DEVICE\_IDENTIFIER\_TYPE\_USB\_GUID**.

The device access protocol is required, because the verifier need have a way  to communciate with the device hardware to get the measurement or do the  challenge/response for the device authentication.

NOTE: We don't use **EFI\_PCI\_IO\_PROTOCOL\_GUID** or **EFI\_USB\_IO\_PROTOCOL\_GUID** here, because we don't want to expose a real protocol. A platform may have driver register a protocol notify function. Installing a real protocol may cause the callback function being executed before the device is authenticated.

**//**

**// Revision The revision to which the DEVICE\_IDENTIFIER interface adhere.**

**//          All future revisions must be backwards compatible.**

**//**

**#define EFI\_DEVICE\_IDENTIFIER\_REVISION 0x00010000**

**//**

**// Device Identifier GUID value**

**//**

**#define EFI\_DEVICE\_IDENTIFIER\_TYPE\_PCI\_GUID \**

**{ \**

**0x2509b2f1, 0xa022, 0x4cca, { 0xaf, 0x70, 0xf9, 0xd3, 0x21, 0xfb, 0x66, 0x49 } \**

**}**

**#define EFI\_DEVICE\_IDENTIFIER\_TYPE\_USB\_GUID \**

**{ \**

**0x7394f350, 0x394d, 0x488c, { 0xbb, 0x75, 0xc, 0xab, 0x7b, 0x12, 0xa, 0xc5 } \**

**}**

**# Special Instructions**

NO