**# Title**: Device Authentication Signature Database

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

**# Document**: UEFI Specification 2.8

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

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

**# Summary of the change**

**[Background]**

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, CXL, etc) 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 based upon a device authentication signature database.

**[Proposal]**

This ECR adds a new device authentication signature database for SPDM device. The device authentication signature database is similar to the UEFI image authentication signature databased used for UEFI secure boot.

For the platform that enables UEFI secure boot, it may optionally enroll the device authentication signature database to authenticate the device besides the UEFI image.

**[Tech Background – SPDM Certificate Chain]**

The SPDM specification defined the Requester and Responder. The Requester could be system firmware. The Responder could be the device to be authenticated.

There are 3 messages are designed for that purpose. See figure 1.

1. GET\_DIGESTS: The system firmware sends this message to get the DIGEST of the SPDM Certificate Chain from the device.
2. GET\_CERTIFICATE: The system firmware sends this message to get the whole SPDM Certificate Chain from the device.
3. CHALLENGE: This is the final step. The system firmware sends this message to challenge the device and expects the device sign the challenge with the device private key.

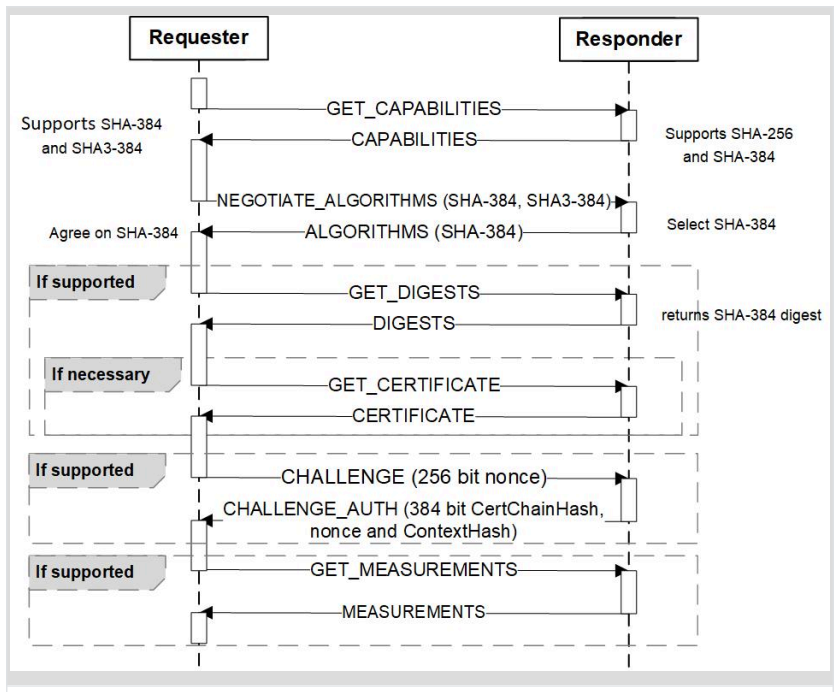
****

Figure 1 – *SPDM Message (Source: SPDM Specification)*

As such, there are at least 2 ways for the verification. See figure 2.

1. The firmware stores the DIGEST of the SPDM Certificate Chain. The firmware must send the GET\_CERTIFICATE message to get the SPDM Certificate Chain then verify the SPDM Certificate Chain. Then the firmware uses the SPDM Certificate Chain to verify the CHALLENG\_AUTH response.
2. The firmware stores the whole SPDM Certificate Chain. The firmware may send GET\_DIFESTS message then verify the SPDM Certificate Chain. Then the firmware uses the SPDM Certificate Chain to verify the CHALLENG\_AUTH response.

Option A consumes small UEFI variable storage but may need more time to get the whole SPDM Certificate Chain at runtime.

Option B consumes large UEFI variable storage but take less time at runtime.

A platform may choose either option based upon the variable storage size and firmware boot time.

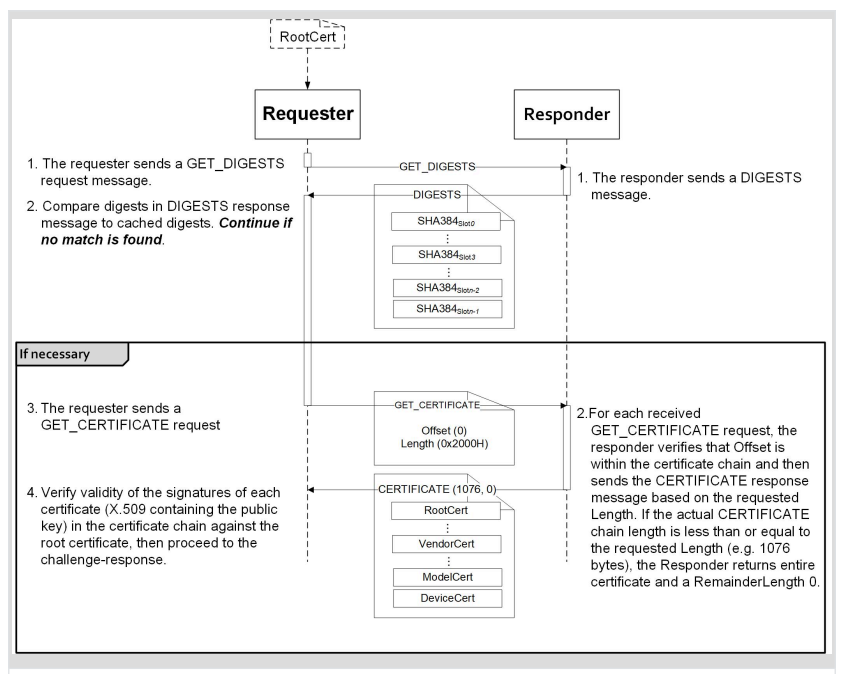


Figure 2 - *Responder authentication: (Source: SPDM Specification)*

The SPDM Certificate Chain format is also defined in SPDM specification.

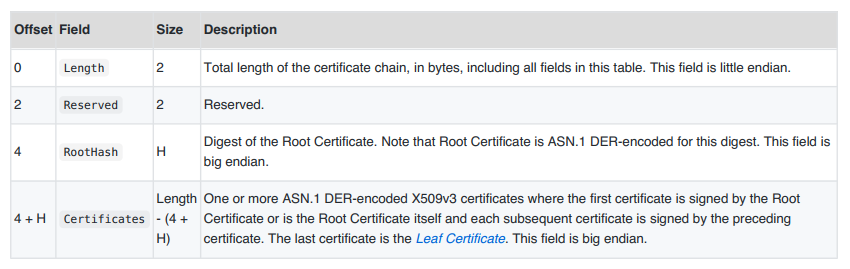


Figure 3 – *SPDM Certificate Chain: (Source: SPDM Specification)*

Reference:

1. UEFI Specification 2.8 - www.uefi.org
2. DMTF SPDM “Security Protocol and Data Model Specification” - https://www.dmtf.org/standards/pmci
3. USB “USB Authentication Specification” - https://www.usb.org/documents
4. PCI-SIG “Component Measurement and Authentication (CMA) ECR” - https://pcisig.com/specifications/review-zone
5. TCG “TCG PC Client Platform firmware Profile Specification” - https://trustedcomputinggroup.org/resource/pc-client-specific-platform-firmware-profile-specification/, https://trustedcomputinggroup.org/wp-content/uploads/TCG\_PCClient\_PFP\_r1p05\_05\_3feb20.pdf
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 image. A platform can use a common UEFI driver to verify all device components on the platform, and corresponding action, such as recovery, which meets the NIST SP 800-193 requirement.
2. The policy/Authority style authentication also enables the possibility on device attestation. The OS or remote third party may use this way to do local attestation or remote attestation for the device. It aligns with the TCG firmware integrity measurement and NIST SP 800-155 requirement.

**# Impact of the change**

1. 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.
2. Add new device signature variable GUID/Name (devdb) in UEFI specification Chapter 32 – Secure Boot and Driver Signing. 32.6.2. UEFI Device Signature Variable GUID and Variable Name.
3. No impact to the existing image authentication signature database.
4. No impact to EFI\_CERT\_EXTERNAL\_MANAGEMENT\_GUID, which is a pseudo-signature which will not facilitate authentication, according to UEFI specification.
5. No impact to SHA1 related definition. SHA1 is considered as unsecure and not used in device authentication.
6. The device revocation or the device forbidden database is out of scope of this proposal. So far, we rely on remote attestation to verify if a device certificate or CA is revoked. The firmware may define its own policy to validate the certificate chain based upon its own knowledge.
7. The policy to determine which device should be verified, and which device need not be verified is out of scope of this proposal. It can be defined by the platform firmware.

# Detailed description of the change [normative updates]

**3.3 Globally Defined Variables**

**Table 14. Global Variables**

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Attribute** | **Description** |
|  |  |  |
| **devdbDefault** | **BS, RT** | The OEM's default device signature store. Should be treated as read-only. |

**32.4.1 Signature Database**

**……**

**#define EFI\_CERT\_X509\_GUID \  
 { 0xa5c059a1, 0x94e4, 0x4aa7, \  
 { 0x87, 0xb5, 0xab, 0x15, 0x5c, 0x2b, 0xf0, 0x72 } };**This identifies a signature based on a DER-encoded X.509 certificate. If the signature is an X.509 certificate then verification of the signature of an image should validate the public key certificate in the image using certificate path verification, up to this X.509 certificate as a trusted root. If the signature is in a device signature variable, this signature is a root certificate authority (CA) certificate for the device. If the Secure Protocol and Data Model (SPDM) is used to authenticate the device, the firmware should send GET\_CERTIFICATE message to get the whole certificate chain, or send GET\_DIGEST message to get the digest of the certificate chain and compare it with local cached certificate chain. Then the firmware should verify the entire SPDM define certificate chain with this root CA. Finally the firmware should send SPDM CHALLENGE message and verify the digital signature of the SPDM CHALLENGE\_AUTH response with the leaf certificate in the verified certificate chain. The *SignatureHeader* size shall always be 0. The *SignatureSize* may vary but shall always be 16 (size of the *SignatureOwner* component) + the size of the certificate itself.  
***Note:*** *This means that each certificate will normally be in a separate* **EFI\_SIGNATURE\_LIST***.*

**……**

**#define EFI\_CERT\_X509\_SHA256\_GUID \  
 { 0x3bd2a492, 0x96c0, 0x4079, \  
 { 0xb4, 0x20, 0xfc, 0xf9, 0x8e, 0xf1, 0x03, 0xed } };  
Prototype  
 #pragma pack(1)  
 typedef struct \_EFI\_CERT\_X509\_SHA256 {  
 EFI\_SHA256\_HASH** *ToBeSignedHash***;  
 EFI\_TIME TimeOfRevocation;  
 } EFI\_CERT\_X509\_SHA256;  
 #pragma pack()  
Members** *ToBeSignedHash* The SHA256 hash of an X.509 certificate’s To-Be-Signed contents.  
 *TimeOfRevocation* The time that the certificate shall be considered to be revoked.  
This identifies a signature containing the SHA256 hash of an X.509 certificate’s To-Be-Signed contents,  
and a time of revocation. If the signature is in a device signature variable, this signature is a SHA256 hash of a root certificate authority (CA) certificate for the device. If the Secure Protocol and Data Model (SPDM) is used to authenticate the device, the firmware should send GET\_CERTIFICATE message to get the whole certificate chain, or send GET\_DIGEST message to get the digest of the certificate chain and compare it with local cached certificate chain. Then the firmware should verify the entire SPDM define certificate chain with this root CA digest. Finally, the firmware should send SPDM CHALLENGE message and verify the digital signature of the SPDM CHALLENGE\_AUTH response with the leaf certificate in the verified certificate chain. The *SignatureHeader* size shall always be 0. The *SignatureSize* shall always be 16 (size of the *SignatureOwner* component) + 48 bytes for an **EFI\_CERT\_X509\_SHA256** structure. If the *TimeOfRevocation* is non-zero, the certificate should be considered to be revoked from that time and onwards, and otherwise the certificate shall be considered to always be revoked.

**#define EFI\_CERT\_X509\_SHA384\_GUID \  
 { 0x7076876e, 0x80c2, 0x4ee6, \  
 { 0xaa, 0xd2, 0x28, 0xb3, 0x49, 0xa6, 0x86, 0x5b } };  
Prototype  
 #pragma pack(1)  
 typedef struct \_EFI\_CERT\_X509\_SHA384 {  
 EFI\_SHA384\_HASH** *ToBeSignedHash***;  
 EFI\_TIME TimeOfRevocation;  
 } EFI\_CERT\_X509\_SHA384;  
 #pragma pack()  
Members** *ToBeSignedHash* The SHA384 hash of an X.509 certificate’s To-Be-Signed contents.  
 *TimeOfRevocation* The time that the certificate shall be considered to be revoked.  
This identifies a signature containing the SHA384 hash of an X.509 certificate’s To-Be-Signed contents,  
and a time of revocation. If the signature is in a device signature variable, this signature is a SHA384 hash of a root certificate authority (CA) certificate for the device. If the Secure Protocol and Data Model (SPDM) is used to authenticate the device, the firmware should send GET\_CERTIFICATE message to get the whole certificate chain, or send GET\_DIGEST message to get the digest of the certificate chain and compare it with local cached certificate chain. Then the firmware should verify the entire SPDM define certificate chain with this root CA digest. Finally, the firmware should send SPDM CHALLENGE message and verify the digital signature of the SPDM CHALLENGE\_AUTH response with the leaf certificate in the verified certificate chain. The *SignatureHeader* size shall always be 0. The *SignatureSize* shall always be 16 (size of the *SignatureOwner* component) + 64 bytes for an **EFI\_CERT\_X509\_SHA384** structure. If the *TimeOfRevocation* is non-zero, the certificate should be considered to be revoked from that time and onwards, and otherwise the certificate shall be considered to always be revoked.

**#define EFI\_CERT\_X509\_SHA512\_GUID \  
 { 0x446dbf63, 0x2502, 0x4cda, \  
 { 0xbc, 0xfa, 0x24, 0x65, 0xd2, 0xb0, 0xfe, 0x9d } };  
Prototype  
 #pragma pack(1)  
 typedef struct \_EFI\_CERT\_X509\_SHA512 {  
 EFI\_SHA512\_HASH** *ToBeSignedHash***;  
 EFI\_TIME TimeOfRevocation;  
 } EFI\_CERT\_X509\_SHA512;  
 #pragma pack()  
Members** *ToBeSignedHash* The SHA512 hash of an X.509 certificate’s To-Be-Signed contents.  
 *TimeOfRevocation* The time that the certificate shall be considered to be revoked.  
This identifies a signature containing the SHA512 hash of an X.509 certificate’s To-Be-Signed contents,  
and a time of revocation. The *SignatureHeader* size shall always be 0. If the signature is in a device signature variable, this signature is a SHA512 hash of a root certificate authority (CA) certificate for the device. If the Secure Protocol and Data Model (SPDM) is used to authenticate the device, the firmware should send GET\_CERTIFICATE message to get the whole certificate chain, or send GET\_DIGEST message to get the digest of the certificate chain and compare it with local cached certificate chain. Then the firmware should verify the entire SPDM define certificate chain with this root CA digest. Finally, the firmware should send SPDM CHALLENGE message and verify the digital signature of the SPDM CHALLENGE\_AUTH response with the leaf certificate in the verified certificate chain. The *SignatureSize* shall always be 16 (size of the *SignatureOwner* component) + 80 bytes for an **EFI\_CERT\_X509\_SHA512** structure. If the *TimeOfRevocation* is non-zero, the certificate should be considered to be revoked from that time and onwards, and otherwise the certificate shall be considered to always be revoked.

32.6 Device Authentication

32.6.1 Overview

This section describes a way to use the platform ownership model to authenticate a platform device during platform boot.

The platform firmware need maintain a device signature database (devdb), which includes a list of the device root CA certificate. The root CA certificate is used to authenticate the device. The device signature database is a single UEFI authenticated variable.

A device root CA certificate in the device signature database may be revoked. In this case, the platform firmware should update the device signature database to remove the old certificate and add a new certificate.

During the system boot, when a bus driver discovers a device, it follows below steps:  
• The bus driver may consult a policy to see if it need authenticate this device. The policy is platform specific. It could be all external devices, all PCI devices, devices attached on certain ports, etc.  
• The bus driver gets the device identity information, such as a device certificate or certificate chain. For example, the Secure Protocol and Data Model (SPDM) GET\_CERTIFICATE command.  
• The bus driver verifies if the certificate or certificate chain is signed by any root CA certificate in the device signature database.   
• The bus driver generates a nonce and uses a challenge/response protocol to verify if the device owns the private key associated with the device certificate. For example, the SPDM CHALLENGE command.   
• After the bus driver passes all verification for the device, the bus driver then enables the device on the UEFI firmware environment. For example, a PCI bus driver will assign bus number, allocate PCI IO/MMIO bar, and install EFI\_PCI\_IO\_PROTOCOL for the PCI device.   
• If any verification fails, the bus driver ignores this device and notifies the platform. The platform firmware may take a platform specific action for the device or the platform. For example, a platform may ignore the device. A platform may disconnect or disable the PCI device. Or a platform may reboot the system or halt the system.

The device authentication flow only verifies the identity of the device and ensure it is a known device. But it does not verify if a device contains the latest certificate or if the device has the latest firmware.  
• A device certificate that is issued by one of the root CA certificates in the device signature may be revoked. The device signature database does not need to be updated. This can be detected by the attestation. For example, if the platform enables trusted boot flow and the platform firmware extends the device certificate to the trust platform module (TPM) platform configuration register (PCR). A verifier can get the device certificate and check the known certificate revocation list (CRL) to see if it is revoked.  
• A device may include an old version firmware. It is not related to the device signature database. This is also be detected by the attestation. For example, the platform firmware may extend the device firmware measurement to TPM PCR. A verifier can get the device firmware information and compare it with the known good device integrity measurement.  
• In both above cases, a platform may define its own policy to perform more verification. For example, a platform may enroll a small set of known revoked certificate. Or a platform may enroll the minimal secure version number for some specific devices.

32.6.2 Authorized User

An *authorized user* (for the purposes of UEFI device authentication) is one who possesses a platform key (PKpriv). This key is used to sign updates to the device signature databases.

32.6.3 Device Signature Database Update

The Authorized device signature databases are stored as UEFI authenticated variables (see Variable Services in Section 8.1.1) with the GUID **EFI\_DEVICE\_SECURITY\_DATABASE\_GUID.**

These authenticated UEFI variables that store the device signature databases (devdb) can always be read but can only be written if:  
• The platform is in user mode and the provided variable data is signed with the private half of the platform private key (PKpriv);  
or if  
• The platform is in setup mode (in this case the variables can be written without a signature  
validation, but the **SetVariable()** call needs to be formatted in accordance with the  
procedure for authenticated variables in Section 8.2.1)

The platform vendor may provide a default set of entries for the Signature Database in the devdbDefault variable described in Section 3.3.

The flow to update the device signature database (devdb) is exactly same as the flow to update the image signature databases, which is described in Section 32.5.3.

32.7.2 UEFI Device Signature Variable GUID and Variable Name

**Summary**

Constants used for UEFI device signature database variable access.

**Prototype**

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

**{0xb9c2b4f4, 0xbf5f, 0x462d, 0x8a, 0xdf, 0xc5, 0xc7, 0xa, 0xc3, 0x5d, 0xad}**

**#define EFI\_DEVICE\_SECURITY\_DATABASE L”devdb”**

**Parameters**

• This GUID and name are used when calling the EFI Runtime Services **GetVariable()** and  
**SetVariable()**.

• The **EFI\_DEVICE\_SECURITY\_DATABASE\_GUID** and **EFI\_DEVICE\_SECURITY\_DATABASE** are  
used to retrieve and change the authorized device signature database.

• Firmware shall support the **EFI\_VARIABLE\_APPEND\_WRITE** flag (see Section 8.1.1) for the  
UEFI device signature database variables.  
• The device signature database variable dbdev must be stored in tamper-resistant nonvolatile storage.

**# Special Instructions**

NO