**# 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.

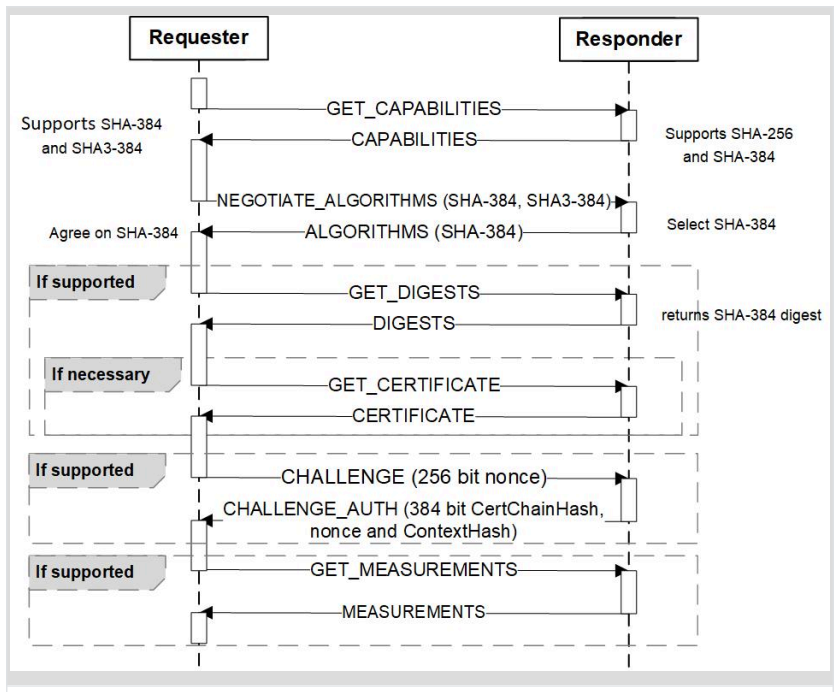
****

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.

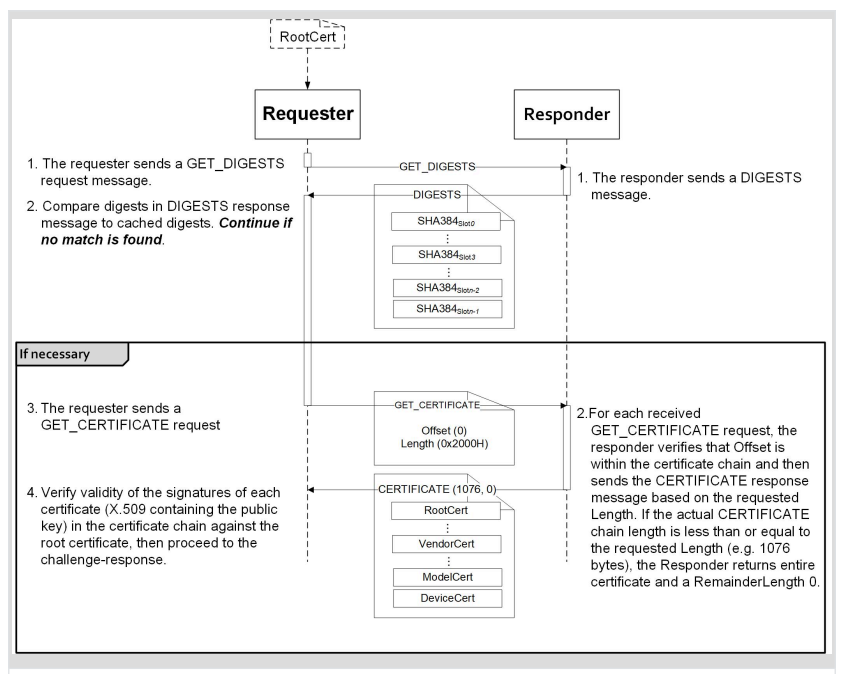


Figure 2 - *Responder authentication: (Source: SPDM Specification)*

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

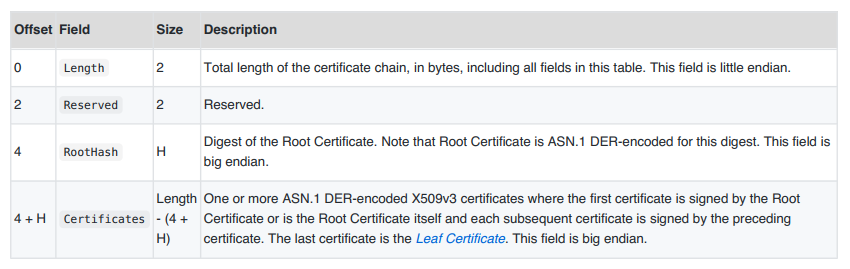


Figure 3 – *SPDM Certificate Chain: (Source: SPDM Specification)*

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. 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 EFI\_CERT\_EXTERNAL\_MANAGEMENT\_GUID, which is a pseudo-signature which will not facilitate authentication, according to UEFI specification.
6. No impact to SHA1 related definition. SHA1 is considered as unsecure and not used in device authentication.

# Detailed description of the change [normative updates]

**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 verify the entire SPDM define certificate chain with this root CA. Then 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 verify the entire SPDM define certificate chain with this root CA digest. Then 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 verify the entire SPDM define certificate chain with this root CA digest. Then 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 verify the entire SPDM define certificate chain with this root CA digest. Then 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.

**#define EFI\_CERT\_X509\_CERT\_CHAIN\_GUID \  
 { 0xa4ba9c9e, 0x1574, 0x4a2e, \  
 { 0x87, 0xc1, 0xc9, 0xbb, 0x87, 0x87, 0x66, 0xe6 } };**

This identifies a signature based on a DER-encoded X.509 certificate chain. This signature could be used in the device authentication. If the Secure Protocol and Data Model (SPDM) is used to authenticate the device, the firmware should send SPDM GET\_DIGESTS message to verify the digest of the SPDM define certificate chain or send SPDM GET\_CERTIFICATE message to verify the entire SPDM define certificate chain. Then the firmware should send SPDM CHALLENGE message and verify the digital signature of the SPDM CHALLENGE\_AUTH response with the leaf certificate in this 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 SPDM defined certificate chain itself.

**#define EFI\_CERT\_X509\_CERT\_CHAIN\_SHA256\_GUID \  
 { 0x14749a83, 0x5c7, 0x4f79, \  
 { 0xab, 0xaf, 0x75, 0x92, 0x71, 0x73, 0xf7, 0x43 } };  
Prototype  
 #pragma pack(1)  
 typedef struct {  
 EFI\_SHA256\_HASH** *Hash***;  
 EFI\_TIME** *TimeOfRevocation***;  
 } EFI\_CERT\_X509\_CERT\_CHAIN\_SHA256;  
 #pragma pack()  
Members** *Hash* The SHA256 hash of the DER-encoded X509 certificate chain.  
 *TimeOfRevocation* The time that the certificate chain shall be considered to be revoked.

This identifies a signature containing a SHA256 hash for the DER-encoded X509 certificate chain. This signature could be used in the device authentication. If the Secure Protocol and Data Model (SPDM) is used to authenticate the device, the firmware should send SPDM GET\_CERTIFICATE message to verify the entire SPDM define certificate chain with the hash of the certificate chain and the revocation time. Then 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\_CERT\_CHAIN\_SHA256. If the *TimeOfRevocation* is non-zero, the certificate chain should be considered to be revoked from that time and onwards, and otherwise the certificate chain shall be considered to always be revoked.

**#define EFI\_CERT\_X509\_CERT\_CHAIN\_SHA384\_GUID \  
 { 0x3c85befc, 0xf863, 0x4db2, \  
 { 0x8d, 0x6a, 0xaf, 0x73, 0x25, 0x4, 0x93, 0x9a } };  
Prototype  
 #pragma pack(1)  
 typedef struct {  
 EFI\_SHA384\_HASH** *Hash***;  
 EFI\_TIME** *TimeOfRevocation***;  
 } EFI\_CERT\_X509\_CERT\_CHAIN\_SHA384;  
 #pragma pack()  
Members** *Hash* The SHA384 hash of the DER-encoded X509 certificate chain.  
 *TimeOfRevocation* The time that the certificate chain shall be considered to be revoked.

This identifies a signature containing a SHA384 hash for the DER-encoded X509 certificate chain. This signature could be used in the device authentication. If the Secure Protocol and Data Model (SPDM) is used to authenticate the device, the firmware should send SPDM GET\_CERTIFICATE message to verify the entire SPDM define certificate chain with the hash of the certificate chain and the revocation time. Then 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\_CERT\_CHAIN\_SHA384. If the *TimeOfRevocation* is non-zero, the certificate chain should be considered to be revoked from that time and onwards, and otherwise the certificate chain shall be considered to always be revoked.

**#define EFI\_CERT\_X509\_CERT\_CHAIN\_SHA512\_GUID \  
 { 0x4795db26, 0x2a19, 0x4ae9, \  
 { 0xae, 0x98, 0x4f, 0xdf, 0xfa, 0x89, 0xe8, 0xab } };  
Prototype  
 #pragma pack(1)  
 typedef struct {  
 EFI\_SHA512\_HASH** *Hash***;  
 EFI\_TIME** *TimeOfRevocation***;  
 } EFI\_CERT\_X509\_CERT\_CHAIN\_SHA512;  
 #pragma pack()  
Members** *Hash* The SHA512 hash of the DER-encoded X509 certificate chain.  
 *TimeOfRevocation* The time that the certificate chain shall be considered to be revoked.

This identifies a signature containing a SHA512 hash for the DER-encoded X509 certificate chain. This signature could be used in the device authentication. If the Secure Protocol and Data Model (SPDM) is used to authenticate the device, the firmware should send SPDM GET\_CERTIFICATE message to verify the entire SPDM define certificate chain with the hash of the certificate chain and the revocation time. Then 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) + 80 bytes for an EFI\_CERT\_X509\_CERT\_CHAIN\_SHA512. If the *TimeOfRevocation* is non-zero, the certificate chain should be considered to be revoked from that time and onwards, and otherwise the certificate chain shall be considered to always be revoked.

32.6.2 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”  
#define EFI\_DEVICE\_SECURITY\_DATABASE1 L”devdbx”**

**#define EFI\_DEVICE\_SECURITY\_DATABASE2 L”devdbt”**

**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.

• The **EFI\_DEVICE\_SECURITY\_DATABASE\_GUID** and **EFI\_DEVICE\_SECURITY\_DATABASE1** are  
used to retrieve and change the forbidden device signature database.

• The **EFI\_DEVICE\_SECURITY\_DATABASE\_GUID** and **EFI\_DEVICE\_SECURITY\_DATABASE2** are  
used to retrieve and change the authorized timestamp signature database.

**# Special Instructions**

NO