**# Title**: UEFI Crypto Agile – remove hardcoded algorithm

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

**# Document**: UEFI Specification Version 2.9

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

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

**# Summary of the change**

**[Background]**

Current UEFI specification hardcoded and mandated some crypto algorithms, such as SHA256 and RSA2048. It might become an issue when the industry has stronger crypto algorithm requirement. For example, the minimal crypto algorithm requirement in Commercial National Security Algorithm Suite (CNSA Suite Guide) is SHA384 + RSAPSS\_3072/ECDSA\_P384. Some vendors are looking for post-quantum cryptography (PQC) algorithm recommended in NIST SP 800-208, such as LMS or XMSS.

Other industry standards already embrace the crypto agile to support multiple algorithms to allow the standard to be used in different environment.

**[Proposal]**

We discussed in USST team. We reached the consensus to move the crypto algorithm requirement from UEFI or PI specification to a standalone document – we name “UEFI Cryptographic Algorithm Guideline”. (Final doc name is TBD)

UEFI forum may provide recommendation and align with crypto guideline from other organization such as National Institute of Standard and Technology (NIST) Federal Information Processing Standards Publications (FIPS), National Security Agency (NSA) Commercial National Security Algorithm Suite(CNSA).

The UEFI or PI specification can just define the mechanism. The crypto algorithm selection is a policy in “UEFI Cryptographic Algorithm Guideline”.

**# Benefits of the change**

1. No UEFI specification change is required, if a platform wants to use a new crypto algorithm.
2. No UEFI specification change is required, if the industry defines new crypto requirement.

**# Impact of the change**

1. Move all hardcoded crypto algorithms from UEFI or PI specification to a standalone document as first step. – No impact to existing platform.
2. Claim multiple algorithms support.

Topic not covered:

1. “UEFI Cryptographic Algorithm Guideline”. That will be covered in another ECR.
2. Define a crypto algorithm reporting mechanism. That will be covered in another ECR.

**Reference:**

1. UEFI Specification 2.9 - <https://uefi.org/specifications>
2. PI Specification 1.7 - <https://uefi.org/specifications>
3. Microsoft secure boot key requirement - [https://docs.microsoft.com/en-us/windows-hardware/manufacture/desktop/windows-secure-boot-key-creation-and-management-guidance#12-public-key-cryptography](https://docs.microsoft.com/en-us/windows-hardware/manufacture/desktop/windows-secure-boot-key-creation-and-management-guidance)
4. Commercial National Security Algorithm Suite (CNSA Suite Guide) - <https://apps.nsa.gov/iaarchive/programs/iad-initiatives/cnsa-suite.cfm>
5. OCP Secure Boot - [https://docs.google.com/document/d/1Se1Dd-raIZhl\_xV3MnECeuu\_I0nF-keg4kqXyK4k4Wc/edit#heading=h.5z2d7x9gbhk0](https://docs.google.com/document/d/1Se1Dd-raIZhl_xV3MnECeuu_I0nF-keg4kqXyK4k4Wc/edit)
6. TCG – EFI protocol - <https://trustedcomputinggroup.org/resource/tcg-efi-protocol-specification/>
7. IETF – Network TLS 1.3 - <https://datatracker.ietf.org/doc/rfc8446/>
8. DMTF – SPDM (Secure Protocol and Data Model) - <https://www.dmtf.org/sites/default/files/standards/documents/DSP0274_1.1.1.pdf>
9. PCI-SIG – CMA (Component Measurement and Authentication) - <https://pcisig.com/specifications>
10. NIST SP800-208 “**Recommendation for Stateful Hash-Based Signature Schemes**” - <https://csrc.nist.gov/publications/sp800>

**# Detailed description of the change [normative updates]**

ADD means ADD, DELETE means DELETE

**8.2.1 Using the EFI\_VARIABLE\_AUTHENTICATION\_3 descriptor**

The NewCert element must have a CertType of **EFI\_CERT\_TYPE\_PKCS7\_GUID** and the CertData must  
be a DER-encoded PKCS#7 version 1.5 SignedData structure. When creating the SignedData structure, the  
following steps shall be followed:

3. Construct a DER-encoded PKCS #7 version 1.5 SignedData (see [RFC2315]) with the signed  
content as follows:  
a SignedData.version shall be set to 1.  
b SignedData.digestAlgorithms shall contain the digest algorithm used when preparing the  
signature. Only a digest algorithm of SHA-256 is accepted. Multiple digest algorithms are allowed.  
c SignedData.contentInfo.contentType shall be set to id-data.  
d SignedData.contentInfo.content shall be the tbsCertificate data that was signed for the  
new x509 cert.  
e SignedData.certificates shall contain, at a minimum, the signer’s DER-encoded X.509  
certificate. Multiple certificates are allowed.  
f SignedData.crls is optional.  
g SignedData.signerInfos shall be constructed as:  
•SignerInfo.version shall be set to 1.  
•SignerInfo.issuerAndSerial shall be present and as in the signer’s certificate.  
•SignerInfo.authenticatedAttributes shall not be present.  
•SignerInfo.digestEncryptionAlgorithm shall be set to the algorithm used to sign the  
data. Only a digest encryption algorithm of RSA with PKCS #1 v1.5 padding  
(RSASSA\_PKCS1v1\_5). is accepted.  
•SignerInfo.encryptedDigest shall be present.  
•SignerInfo.unauthenticatedAttributes shall not be present.  
•Multiple SignerInfos are allowed.  
4. Set the CertData field to the DER-encoded PKCS#7 SignedData value.

**8.2.2 Using the EFI\_VARIABLE\_AUTHENTICATION\_2 descriptor**

3. Sign the resulting digest using a selected signature scheme (e.g. PKCS #1 v1.5)  
4. Construct a DER-encoded PKCS #7 version 1.5 SignedData (see [RFC2315]) with the signed  
content as follows:  
a SignedData.version shall be set to 1  
b SignedData.digestAlgorithms shall contain the digest algorithm used when preparing the  
signature. Only a digest algorithm of SHA-256 is accepted. Multiple digest algorithms are allowed.  
c SignedData.contentInfo.contentType shall be set to id-data  
d SignedData.contentInfo.content shall be absent (the content is provided in the Data  
parameter to the SetVariable() call)  
e SignedData.certificates shall contain, at a minimum, the signer’s DER-encoded X.509  
certificate. Multiple certificates are allowed.  
f SignedData.crls is optional.  
g SignedData.signerInfos shall be constructed as:  
— SignerInfo.version shall be set to 1  
— SignerInfo.issuerAndSerial shall be present and as in the signer’s certificate —  
SignerInfo.authenticatedAttributes shall not be present.  
— SignerInfo.digestEncryptionAlgorithm shall be set to the algorithm used to sign the data.  
Only a digest encryption algorithm of RSA with PKCS #1 v1.5 padding (RSASSA\_PKCS1v1\_5).  
is accepted.  
— SiginerInfo.encryptedDigest shall be present  
— SignerInfo.unauthenticatedAttributes shall not be present

— Multiple SignerInfos are allowed.

**27.2 EAP Protocol**

EFI\_EAP.SetDesiredAuthMethod()

The **SetDesiredAuthMethod()** function sets the desired EAP authentication method indicated by  
*EapAuthType* for the Port.  
If *EapAuthType* is an invalid EAP authentication type, then **EFI\_INVALID\_PARAMETER** is returned.  
If the EAP authentication method of *EapAuthType* is unsupported, then it will return  
**EFI\_UNSUPPORTED**

The cryptographic strength of **EFI\_EAP\_TYPE\_TLS** shall be at least of hash strength SHA-256 and RSA  
key length of at least 2048 bits.

**32.3 Firmware/OS Key Exchange: creating trust relationships**

Platform Key (PK)  
The platform key establishes a trust relationship between the platform owner and  
the platform firmware. The platform owner enrolls the public half of the key (PKpub)  
into the platform firmware. The platform owner can later use the private half of the  
key (PKpriv) to change platform ownership or to enroll a Key Exchange Key. For UEFI ,  
the recommended Platform Key format is RSA-2048. See “Enrolling The Platform  
Key” and “Clearing The Platform Key” for more information.

**32.5.3.3 Authorization Process**

3. UEFI Image Validation Succeeded …

C. Any entry with **SignatureListType** of **EFI\_CERT\_X509\_GUID**, with **SignatureData**which contains a certificate with the same Issuer, Serial Number, and To-Be-Signed hash  
included in anyy certificate in the signing chain of the signature being verified.  
Multiple signatures are allowed to exist in the binary’s certificate table (as per PE/COFF Section  
“Attribute Certificate Table”). Only one hash or signature is required to be present in *db*in order to pass validation, so long as neither the SHA-256 hash of the binary nor any  
present signature is reflected in dbx.

**37.4 PKCS7 Verify Protocol**

The **EFI\_PKCS7\_VERIFY\_PROTOCOL** is used to verify data signed using PKCS7 structure. PKCS7 is a  
general-purpose cryptographic standard (see references). The PKCS7 data to be verified must be ASN.1  
(DER) encoded. Implementation must support SHA256 as digest algorithm with RSA digest encryption.  
Support of other hash algorithms is optional. See Table 32.

**Table 37-5. Details of Supported Signature Format**

|  |  |
| --- | --- |
| Signature Buffer Format Details |  |
| Encoding | Binary DER |
| ASN.1 root of Embedded Signed Data | ContentInfo with SignedDatacontent type |
| ASN.1 root of Detached Signature | SignedData or ContentInfo with SignedData content type |
| Embedded Data Type | Typically ‘Data’ (1.2.840.113549.1.7.1) or other defined OID type (however caller should not depend upon specialized OID processing during PKCS validation.) |
| Digest (Hash) Algorithm(VerifyBufferfunction) | Support of SHA-256 (2.16.840.1.101.3.4.2.1) is required, other algorithms are optional See [RFC2315] and OID definition by different standard body. For example:  SHA-256**:** (2.16.840.1.101.3.4.2.1)  SHA-384**:** (2.16.840.1.101.3.4.2.2)  SHA-512**:** (2.16.840.1.101.3.4.2.3) |
| Digest Encryption | RSA (1.2.840.113549.1.1.1) See [RFC2315] and OID definition by different standard body. For example:  RSAES-PKCS1-v1\_5: (1.2.840.113549.1.1.1) |
| Certificate validity dates | See TimeStampDbdescription |
| Signature authenticated Attributes | Ignored by function |
| Timestamping | See TimeStampDbdescription |

***Note:*** *Because this function uses hashes and the specification contains a variety of hash choices, you  
should be aware that the check against the RevokedDb list will improperly succeed if the  
signature is revoked using a different hash algorithm. For this reason, you should either cycle  
through all UEFI supported hashes to see if one is forbidden, or rely on a single hash choice*

*only if the UEFI signature authority only signs and revokes with a single hash (at time of  
writing, this hash choice is SHA256).*