**# Title**: PI Support More Crypto Algorithm

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

**# Document**: PI Specification Version 1.7 (<https://uefi.org/sites/default/files/resources/PI_Spec_1_7_A_final_May1.pdf>)

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

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

**# Summary of the change**

**[Background]**

UEFI 2.10 specification adds crypto agile to support more crypto algorithm (<https://bugzilla.tianocore.org/show_bug.cgi?id=3725>).

PI specification signed FV and signed section only supports PKCS7 or RSA2048\_SHA256 cert\_type. Unlike UEFI, the PI implementation did not use the PKCS7 but chose the raw digital signature RSA2048\_SHA256, due to size constrain in PEI phase. As such, we need extend RSA2048\_SHA256 to RSA3072/RSA4096/ECDSA\_P384, etc.

**# Benefits of the change**

Support more cryptography for signed FV and signed section.

Align with the crypto algorithm defined in UEFI 2.10.

Support NSA CNSA 1.0 Compliance

**# Impact of the change**

Reference:

[1] UEFI Specification 2.10, 32.5 Firmware/OS Crypto Algorithm Exchange, <https://uefi.org/specs/UEFI/2.10/32_Secure_Boot_and_Driver_Signing.html?highlight=efi_sha256_hash#firmware-os-crypto-algorithm-exchange>

[2] CNSA 1.0, <https://apps.nsa.gov/iaarchive/programs/iad-initiatives/cnsa-suite.cfm>

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

ADD means ADD, DELETE means DELETE

**PI Specification Vol 3.**

**3.2.1.1 EFI Signed Firmware Volumes**There may be one or more headers with a *FormatType* of value  
**EFI\_FIRMWARE\_CONTENTS\_SIGNED\_GUID.**A *signed firmware volume* is a cryptographic signature across the entire volume. To process the  
contents and verify the integrity of the volume, the  
**EFI\_FIRMWARE\_VOLUME\_EXT\_ENTRY\_GUID\_TYPE** *Data[]*shall contain an instance of  
**WIN\_CERTIFICATE\_UEFI\_GUID** where the *CertType* **=  
EFI\_CERT\_TYPE\_PKCS7\_GUID**or **EFI\_CERT\_TYPE\_RSA2048\_SHA256\_GUID**, **EFI\_CERT\_TYPE\_RSA3072\_SHA384\_GUID**, **EFI\_CERT\_TYPE\_RSA4096\_SHA512\_GUID**, **EFI\_CERT\_TYPE\_RSAPSS3072\_SHA512\_GUID**, **EFI\_CERT\_TYPE\_RSAPSS4096\_SHA512\_GUID**, **EFI\_CERT\_TYPE\_ECDSA\_ECC\_NIST\_P256\_SHA256\_GUID**, **EFI\_CERT\_TYPE\_ECDSA\_ECC\_NIST\_P384\_SHA384\_GUID**.

**Related Definitions**

**#define EFI\_CERT\_TYPE\_RSA3072\_SHA384\_GUID  
{0xdf70754f, 0xcf85, 0x47de, \**

**{0xbb, 0x49, 0xd8, 0x36, 0x1d, 0x4b, 0xf0, 0xd5}}**

**#define EFI\_CERT\_TYPE\_RSA4096\_SHA512\_GUID  
{0xeffa3c4c, 0x2181, 0x4ca2, \**

**{0xa3, 0xe6, 0xad, 0xdb, 0x8b, 0x68, 0x52, 0x98}}**

**#define EFI\_CERT\_TYPE\_RSAPSS3072\_SHA384\_GUID  
{0x1bbe9dda, 0x965f, 0x4b1a, \**

**{0x92, 0xce, 0xe9, 0x4e, 0x19, 0xe5, 0x61, 0x28}}**

**#define EFI\_CERT\_TYPE\_RSAPSS4096\_SHA512\_GUID  
{0xa2743703, 0xf28c, 0x416c, \**

**{0xb2, 0xed, 0xc6, 0x76, 0x27, 0x57, 0x89, 0x90}}**

**#define EFI\_CERT\_TYPE\_ECDSA\_ECC\_NIST\_P256\_SHA256\_GUID  
{0x7c6c3269, 0xc9fb, 0x4762, \**

**{0x81, 0xe1, 0x55, 0x40, 0x21, 0xf7, 0x76, 0x6e}}**

**#define EFI\_CERT\_TYPE\_ECDSA\_ECC\_NIST\_P384\_SHA384\_GUID  
{0x216afe76, 0x18fe, 0x4d68, \**

**{0x83, 0x72, 0x55, 0xbc, 0xf9, 0x9b, 0x6f, 0x65}}**

**typedef struct \_EFI\_CERT\_BLOCK\_RSA\_3072\_SHA384 {  
 EFI\_GUID** *HashType***;  
 UINT8** *PublicKey***[384];  
 UINT8** *Signature***[384];  
} EFI\_CERT\_BLOCK\_RSA\_3072\_SHA384;**

*PublicKey* The RSA exponent e for this structure is 0x10001.

*Signature* This signature block is PKCS 1 version 1.5 formatted.

**typedef struct \_EFI\_CERT\_BLOCK\_RSA\_4096\_SHA512 {  
 EFI\_GUID** *HashType***;  
 UINT8** *PublicKey***[512];  
 UINT8** *Signature***[512];  
} EFI\_CERT\_BLOCK\_RSA\_4096\_SHA512;**

*PublicKey* The RSA exponent e for this structure is 0x10001.

*Signature* This signature block is PKCS 1 version 1.5 formatted.

**typedef struct \_EFI\_CERT\_BLOCK\_ECDSA\_NIST\_P256\_SHA256 {  
 EFI\_GUID** *HashType***;  
 UINT8** *PublicKey***[64];  
 UINT8** *Signature***[64];  
} EFI\_CERT\_BLOCK\_ECDSA\_NIST\_P256\_SHA256;**

*PublicKey* The public key is ECDSA public key (x, y). The bytes order of x and y is big endian. The first 32 bytes are x, the second 32 bytes are y.

*Signature* This signature block is FIPS 186-4 defined ECDSA signature (r, s). The byte order of r and s is big endian. The first 32 bytes are r, the second 32 bytes are s.

**typedef struct \_EFI\_CERT\_BLOCK\_ECDSA\_NIST\_P384\_SHA384 {  
 EFI\_GUID** *HashType***;  
 UINT8** *PublicKey***[96];  
 UINT8** *Signature***[96];  
} EFI\_CERT\_BLOCK\_ECDSA\_NIST\_P384\_SHA384;**

*PublicKey* The public key is ECDSA public key (x, y). The bytes order of x and y is big endian. The first 48 bytes are x, the second 48 bytes are y.

*Signature* This signature block is FIPS 186-4 defined ECDSA signature (r, s). The byte order of r and s is big endian. The first 48 bytes are r, the second 48 bytes are s.

**Description**

The **WIN\_CERTIFICATE\_UEFI\_GUID** certificate type allows new types of certificates to be developed  
for driver authentication without requiring a new certificate type. The *CertType* defines the format of  
the *CertData*, which length is defined by the size of the certificate less the fixed size of the  
**WIN\_CERTIFICATE\_UEFI\_GUID** structure. Besides UEFI defined **EFI\_CERT\_TYPE\_PKCS7\_GUID** or **EFI\_CERT\_TYPE\_RSA2048\_SHA256\_GUID**, this specification adds below *CertType*:  
• If *CertType* is **EFI\_CERT\_TYPE\_RSA3072\_SHA384\_GUID** or **EFI\_CERT\_TYPE\_RSAPSS3072\_SHA384\_GUID** then the structure which follows has the format specified by **EFI\_CERT\_BLOCK\_RSA\_3072\_SHA384**.  
• If *CertType* is **EFI\_CERT\_TYPE\_RSA4096\_SHA512\_GUID** or **EFI\_CERT\_TYPE\_RSAPSS4096\_SHA512\_GUID** then the structure which follows has the format specified by **EFI\_CERT\_BLOCK\_RSA\_4096\_SHA512**.  
• If *CertType* is **EFI\_CERT\_TYPE\_ECDSA\_ECC\_NIST\_P256\_SHA256\_GUID** then the structure which follows has the format specified by **EFI\_CERT\_BLOCK\_ECDSA\_NIST\_P256\_SHA256**.  
• If *CertType* is **EFI\_CERT\_TYPE\_ECDSA\_ECC\_NIST\_P384\_SHA384\_GUID** then the structure which follows has the format specified by **EFI\_CERT\_BLOCK\_ECDSA\_NIST\_P384\_SHA384**.

**3.2.5 Firmware File Section Types**

**…**

**EFI Signed Sections**For **EFI\_GUID\_DEFINED\_SECTION** and **EFI\_GUID\_DEFINED\_SECTION2** there is a  
*SectionDefinitionGuid* of type **EFI\_FIRMWARE\_CONTENTS\_SIGNED\_GUID.**The *GuidSpecificHeaderFields* shall include an entry *SignatureInfo* of type  
**WIN\_CERTIFICATE\_UEFI\_GUID***.***#define EFI\_FIRMWARE\_CONTENTS\_SIGNED\_GUID \  
{ 0xf9d89e8, 0x9259, 0x4f76, \**   
**{ 0xa5, 0xaf, 0xc, 0x89, 0xe3, 0x40, 0x23, 0xdf } }**The *signed section* is an encapsulation section in which the section data is cryptographically signed.  
To process the contents and extract the enclosed section stream, the section data integrity must be  
accessed by evaluating the enclosed data via the cryptographic information in the  
*SignatureInfo*. The *CertType* **= EFI\_CERT\_TYPE\_PKCS7\_GUID** or  
**EFI\_CERT\_TYPE\_RSA2048\_SHA256\_GUID**, **EFI\_CERT\_TYPE\_RSA3072\_SHA384\_GUID**, **EFI\_CERT\_TYPE\_RSA4096\_SHA512\_GUID**, **EFI\_CERT\_TYPE\_RSAPSS3072\_SHA512\_GUID**, **EFI\_CERT\_TYPE\_RSAPSS4096\_SHA512\_GUID**, **EFI\_CERT\_TYPE\_ECDSA\_ECC\_NIST\_P256\_SHA256\_GUID**, **EFI\_CERT\_TYPE\_ECDSA\_ECC\_NIST\_P384\_SHA384\_GUID**.

The signed image is then interpreted as a section stream. **EFI\_GUID\_DEFINED\_SECTION2** is  
used if the section is 16MB or larger.