POST-QUANTUM CRYPTOGRAPHY: UEFI SPECIFICATION UPDATES

Editors:
Jiewen Yao (Intel) and Sean Brogan (Microsoft)

Contributors: UEFI Security Sub-Team (USST): Alex Podgorsky (AMI), Andrei Popov (Microsoft), Bill Munger (Dell), Chow Jeremy (Dell), Collin Parker (Lenovo), Doug Flick (Microsoft), Goutam Hegde (Cisco), Inbal Levi (Microsoft), James Bottomley (Linux Foundation), Jason Fisher (Microsoft), Jordan Geurten (Microsoft), Jordan Rogers (Microsoft), Kevin Davis (Insyde), Lenny Szubowicz (Red Hat), Michael Kinney (Intel), Mike Demeter (Lenovo), Nikolay Kalaichidi (Dell), Patrick Gibbons (Lenovo), Peter Jones (Red Hat), Prasanna Karthik Mutharaju (Microsoft), Rick Bramley (HP, Inc), Sachin Ganesh (AMI), Scott Shell (Microsoft), Sochi Ogbuanya (Microsoft), Srini Narayana (AMI), Stuart Yoder (ARM), Terry Lee (HP Enterprise), Tim Hoppen (Phoenix), Tonry Richard (Dell), Yi Li (Intel)

Publish Date: April 29, 2026

Table of Contents

1. Introduction

The advent of quantum computing poses a significant threat to the cryptographic algorithms that underpin the security of modern computing platforms. Algorithm such as RSA2048, which is widely used in UEFI Secure Boot and authenticated variable, is vulnerable to quantum attacks.

Over the last year, the UEFI Secure Sub Team (USST), a sub-team of the UEFI Specification Working Group (USWG), has met to define a path forward for several current and emerging challenges:

- Adoption of post-quantum cryptography (PQC) for UEFI features and platforms.
- Impacts of the 2011 UEFI Secure Boot key expiration on in-market PCs.
- Review of the UEFI security features and common implementation/usage patterns, and the depreciation of unused and unnecessary complexity.

UEFI Secure Boot has been a primary focus of this effort. For the purposes of this paper, "UEFI Secure Boot" refers to the UEFI feature that defines:

- How binaries are authenticated (code signing verification).
- How execution is authorized using the allow list (db).
- How execution is revoked or blocked using the revoke list (dbx).
- How updates to PK, KEK, db, and dbx are authenticated and authorized.

Secure Boot is fully specified in the UEFI specification because it is critical to interoperability between platform suppliers, OS vendors, and application developers. The specification provides the contract that enables broad ecosystem adoption. While the UEFI specification also defines cryptographic details for features such as Capsule Update, the Firmware Management Protocol, TLS, and HTTPS Boot, those features typically have fewer cross-vendor compatibility constraints and therefore do not require the same level of coordination.

The working group used the following principles to guide these proposals:

- The threat landscape evolves quickly, while UEFI specifications and deployed firmware are slower to change and hard to remediate at scale.
- Security requirements vary by geography and regulation; "PQC readiness" can mean different things in different contexts.
- PQC algorithms are still maturing, so we should expect iteration and plan for change.

NOTE: This whitepaper consolidates proposed specification changes for additional feedback and to validate implementation feasibility. These proposals are not yet approved and may change or be rejected. **Do not ship UEFI-based products implementing this document; any final requirements will be defined by the published UEFI specification.** By sharing this publicly, the UEFI Forum aims to surface implementation and compatibility risks and gather broader input before finalizing updates.

2. PQC Background

PQC Compliance and Timeline

The National Institute of Standards and Technology (NIST) has finalized several PQC standards:

- **FIPS 203** — ML-KEM (Module Lattice-based Key Encapsulation Mechanism)
- **FIPS 204** — ML-DSA (Module Lattice-based Digital Signature Algorithm)
- **FIPS 205** — SLH-DSA (Stateless Hash-based Digital Signature Algorithm)

Additional guidance includes NIST SP 800-227 (KEM Recommendation) and NIST IR 8547 (PQC Transition). Government agencies such as the NSA (CNSA 2.0), the EU, UK NCSC, Germany BSI, and France ANSSI have published migration timelines and recommendations for PQC adoption.

UEFI Crypto Usage

UEFI relies on cryptographic operations in several key areas:

- **Secure Boot image verification** — Verifying the digital signatures of UEFI images (OS loaders, drivers, option ROMs) using the authorized signature database (db).
- **Secure Boot authorization** — Managing the Platform Key (PK), Key Exchange Key (KEK), and signature databases (db/dbx/dbt/dbr).
- **Authenticated variable updates** — Using PKCS#7 signatures to authenticate updates to PK, KEK, and signature databases.
- **Image revocation** — Managing the forbidden signature database (dbx) to block compromised images.
- **Device firmware update** — Firmware capsule update via ESRT and FMP.

All of these areas are impacted by PQC and require specification updates to ensure crypto agility.

3. UEFI Specification PQC Update Plan

General Design Principles

The UEFI PQC update plan follows several guiding the design principles:

- **Make infrastructure ready for PQC (crypto agility)** — Ensure the specification supports PQC algorithms without mandating specific ones.
- **Reuse PQC crypto defined by the industry** — Leverage standards from NIST, IETF, and other bodies (e.g., ML-DSA in RFC 9881/9882, SLH-DSA in RFC 9909/9814, composite signatures in draft-ietf-lamps-pq-composite-sigs).
- **Support extension for the future** — Allow the specification to accommodate future algorithms and combinations.

Out of scope:

- Mandating any specific PQC algorithm.
- Defining new PQC crypto primitives or hybrid combinations.
- Defining measurement requirements (deferred to TCG).

The UEFI.next specification release will include the updates required for PQC enablement.

A. New Features

A1. Allow Multiple Signature Verification (Fix DBX Policy)

Problem Statement: During the PQC transition, UEFI images may carry multiple signatures (e.g., both a traditional RSA signature and a PQC ML-DSA signature) to ensure backward compatibility and forward security. However, the current specification states that verification must not succeed if "*neither* the hash of the binary *nor any* present signature is reflected in *dbx*." This language blocks the expected multiple signature usage — if one algorithm is revoked, the entire image is rejected even if another valid signature exists.

Proposal:

- Clarify the dbx policy with a revised verification order:
- If the hash of the image is in DBX → **FAIL**
- Else if the hash of the image is in DB → **PASS**
- Else if one of the image's signatures is in DB but not in DBX → **PASS**
- Else → **FAIL**

UEFI Spec Change: In Chapter 32 (Secure Boot and Driver Signing), Section "Authorization Process", the existing text "Only one hash or signature is required to be present in *db* in order to pass validation, so long as neither the hash of the binary nor any present signature is reflected in dbx" is replaced with a prescriptive validation order: - A. If the hash of the binary is in *dbx*, the image shall fail. - B. Else if the hash of the binary is in *db*, the image shall pass. - C. Else if one of the signatures is in *db* and is not in *dbx*, the image shall pass. - D. Else the image shall fail.

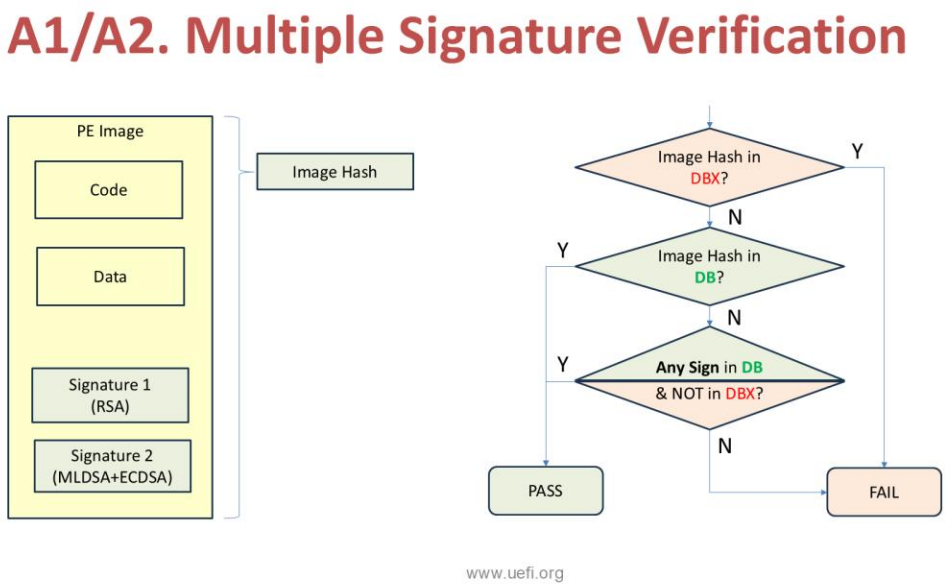

Figure 1: Multiple Signature Verification Flow

A2. Mandate the Order of Multiple Signature Verification

Problem Statement: When a UEFI image carries multiple signatures, the specification does not define the order in which they should be evaluated. This ambiguity can lead to inconsistent behavior across implementations.

Proposal:

- Mandate the order of multiple signature verification: top-down based on the signature order in the UEFI image.

UEFI Spec Change: In Chapter 32, Section "Authorization Process", new text is added: "If an image includes multiple signatures, the firmware shall evaluate each signature sequentially in the order they appear in the image's certificate table. For each signature, the firmware shall verify it against *db* and *dbx*. If a signature is accepted, the image is accepted and the remaining signatures are not evaluated. If a signature is not accepted, the firmware shall proceed to evaluate the next signature. If all signatures have been evaluated and none is accepted, the image is rejected."

A3. Add TBSCertificate Hash for DB

Problem Statement: PQC certificates are significantly larger than traditional certificates. Storing full PQC certificates in DB consumes large amounts of flash space.

Proposal:

- Add TBSCertificate (To-Be-Signed) hash support for DB using `EFI_CERT_X509_SHA256/SHA384/SHA512_GUID`.

- The final database may include:
- **(DB/DBX)** Hash of image — `EFI_CERT_SHA256/SHA384/SHA512_GUID`
- **(DB/DBX)** TBSCertificate hash — `EFI_CERT_X509_SHA256/SHA384/SHA512_GUID`
- **(DB only)** Full certificate — `EFI_CERT_X509_GUID`

UEFI Spec Change: In Chapter 32, multiple changes: - Section "Signature Database" — For `EFI_CERT_X509_SHA256/384/512_GUID` and `EFI_CERT_X509_SM3_GUID` descriptions, add: "The *TimeOfRevocation* field is only valid for the forbidden signature database (dbx), and it should be 0 when produced and be ignored when consumed in other cases." - Section "Authorization Process" — The DB matching criteria is expanded to explicitly list three entry types: (A) hash of image, (B) TBSCertificate hash, and (C) full certificate.

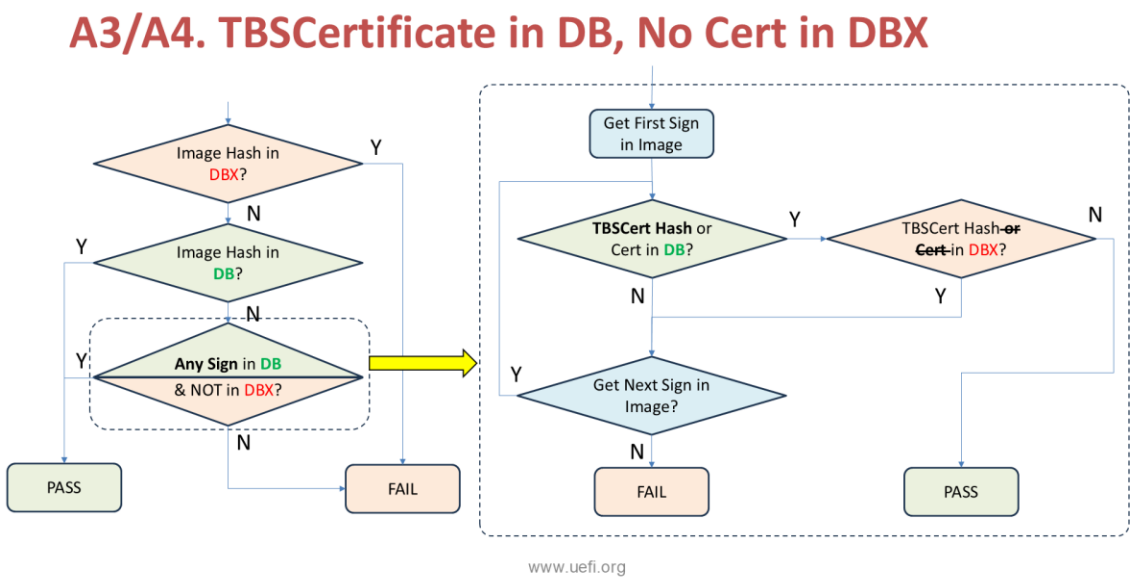

Figure 2: TBSCertificate Verification Flow

A4. Mandate TBSCertificate or Hash for DBX and Remove Certificate from DBX

Problem Statement: Storing full PQC certificates in DBX is costly and creates problems when revoking multiple PQC certificates at once (due to algorithm checks).

Proposal:

- Remove full certificate (`EFI_CERT_X509_GUID`) usage from DBX.
- The final database may include:
- **(DB/DBX)** Hash of image — `EFI_CERT_SHA256/SHA384/SHA512_GUID`
- **(DB/DBX)** TBSCertificate hash — `EFI_CERT_X509_SHA256/SHA384/SHA512_GUID`

- **(DB only)** Full certificate — `EFI_CERT_X509_GUID`

UEFI Spec Change: In Chapter 32, after the "Authorization Process" section, a deprecation note is added: "**DEPRECATED: EFI_CERT_X509_GUID for DBX** — The use of the `EFI_CERT_X509_GUID` signature type for *dbx* is deprecated and should no longer be used. It is recommended to use To-Be-Signed hash entries (`EFI_CERT_X509_SHA256/SHA384/SHA512`) in *dbx* instead."

A5. EFI Crypto Indicator Table (ECIT)

Problem Statement: An OS loader or application may need to determine whether the UEFI firmware is PQC-ready. Currently, there is no mechanism for firmware to report which cryptographic algorithms it supports.

Proposal:

- Define a new EFI Crypto Indicator Table, published as a UEFI Configuration Table and an ACPI table.
- The table reports the cryptographic algorithms supported by the UEFI firmware across different usage categories:
  - **UEFI Image Verification (DB - PKCS7):** e.g., sha256WithRSAEncryption, ecdsa-with-SHA256, id-ml-dsa-44/65/87
  - **UEFI Secure Boot Authorization (DB/KEK/PK):** e.g., EFI_CERT_X509, EFI_CERT_SHA256/384/512, EFI_CERT_X509_SHA256/384/512
  - **UEFI Authenticated Variable (PK/KEK - PKCS7):** e.g., sha256WithRSAEncryption, ecdsa-with-SHA256, id-ml-dsa-44/65/87
  - **UEFI Image Revocation (DBX):** Supported signature types
  - **UEFI Device Firmware Update (ESRT)** and **System Firmware Update (FMP)**
- Allow OEM/ODM/IBV extensions for vendor-specific crypto usage reporting.

UEFI Spec Change: A new section "EFI Crypto Indicator Table" is added to Chapter 37 (Secure Technologies) defining: - `EFI_CRYPTO_INDICATOR_TABLE` — An ACPI-compatible Configuration Table with signature "ECIT", containing an array of `EFI_CRYPTO_INDICATOR_ENTRY` structures. - Each entry uses a Feature Identifier GUID and opaque `EntryData` to declare supported algorithms. - Defined feature entries include: Image Verification (PKCS7 OIDs), Secure Boot Authorization (SignatureList types for db/KEK/PK), Secure Boot Servicing Authorization (SignatureList types for PK/KEK), Image Revocation (SignatureList types for dbx), Authenticated Variable (PKCS7 OIDs), System Firmware Update, and ESRT Device Firmware Update. - Extensible for OEM/ODM/IBV-specific features via custom GUIDs. - Published as an `EFI_CONFIGURATION_TABLE`.

Data Structure:

Configuration Table GUID:

```
// {1768b8b1-1605-401a-bc49-d612d2b98c4e}
#define EFI_CRYPTO_INDICATOR_TABLE_GUID \
```

```
  {0x1768b8b1, 0x1605, 0x401a, \
  {0xbc, 0x49, 0xd6, 0x12, 0xd2, 0xb9, 0x8c, 0x4e}}
```

Table Structure:

```
#define EFI_CRYPTO_INDICATOR_TABLE_VERSION 1

typedef struct {
  char     Signature[4];       // "ECIT"
  UINT32   Length;              // Length of entire table
  UINT8    Version;             //
EFI_CRYPTO_INDICATOR_TABLE_VERSION
  UINT8    Checksum;            // Checksum of entire table
  char     OemId[6];
  char     OemTableId[8];
  UINT32   OemRevision;
  UINT32   CreatorID;
  UINT32   CreatorRevision;
  // ECIT-specific fields
  UINT8    NumberOfEntries;
  UINT8    Reserved[3];         // Padding, should be set to zero
  EFI_CRYPTO_INDICATOR_ENTRY Entries[];
} EFI_CRYPTO_INDICATOR_TABLE;
```

Entry Structure:

```
typedef struct {
  EFI_GUID FeatureIdentifier;
  UINT16   EntryLength;  // sizeof(EFI_CRYPTO_INDICATOR_ENTRY) +
sizeof(EntryData)
  UINT8    Reserved[6];  // Padding, should be set to zero
  UINT8    EntryData[];
} EFI_CRYPTO_INDICATOR_ENTRY;
```

Feature Identifier GUIDs and Entry Data Types:

Feature	**GUID**	**EntryData Type**
Image Verification (cert in db)	`EFI_ECIT_FEATURE_IMAGE_VERIFICATION_GUID`	CSV string of supported algorithm OIDs (e.g., `sha256WithRSAEncryption,id-ml-`

		dsa-65)
Secure Boot Authorization (db)	EFI_ECIT_FEATURE_SECURE_BOOT_AUTHORIZATION_GUID	Array of EFI_GUID — supported EFI_SIGNATURE_LIST types
Image Revocation (dbx)	EFI_ECIT_FEATURE_IMAGE_REVOCATION_GUID	Array of EFI_GUID — supported EFI_SIGNATURE_LIST types
Secure Boot Servicing Authorization (PK/KEK)	EFI_ECIT_FEATURE_SECURE_BOOT_SERVICING_AUTHORIZATION_GUID	Array of EFI_GUID — supported EFI_SIGNATURE_LIST types
Authenticated Variable (cert in KEK/PK)	EFI_ECIT_FEATURE_AUTHENTICATED_VARIABLE_GUID	CSV string of supported algorithm OIDs
System Firmware Update	EFI_ECIT_FEATURE_SYSTEM_FIRMWARE_UPDATE_GUID	CSV string of supported algorithm OIDs
ESRT Device Firmware Update	EFI_ECIT_FEATURE_ESRT_FIRMWARE_UPDATE_GUID	EFI_GUID Esrt_Guid + CSV string of supported algorithm OIDs

The Image Verification, Secure Boot Authorization, Secure Boot Servicing Authorization, Image Revocation, and Authenticated Variable feature entries are *required* for any system that supports UEFI Secure Boot. Other features can be defined by other specifications or vendors using custom GUIDs.

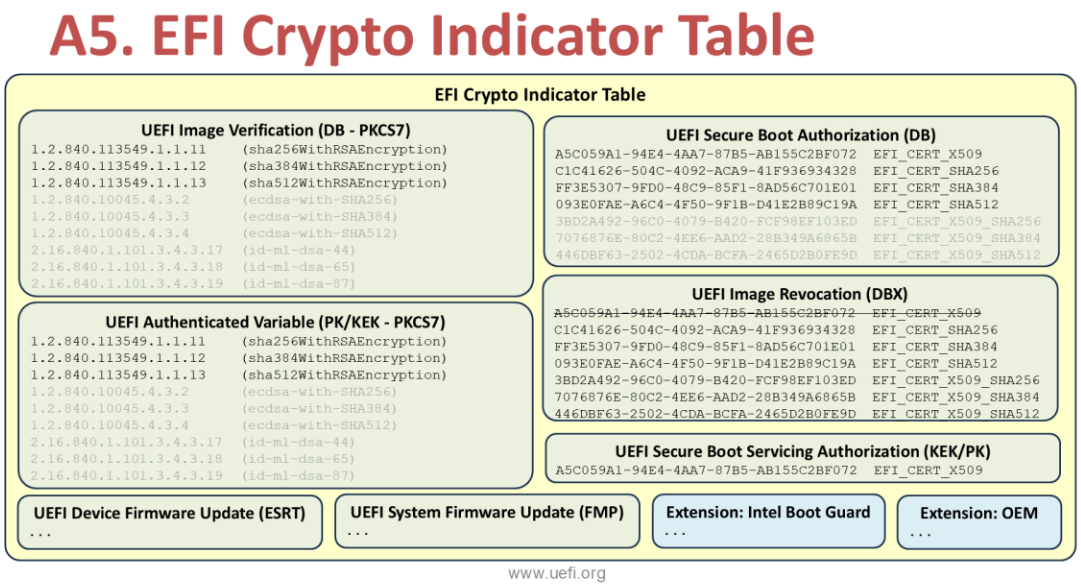

Figure 3: EFI Crypto Indicator Table Structure

A6. Algorithm Check for PK/KEK/db/dbx Enrollment

Problem Statement: Currently, there is no algorithm validation when PK, KEK, db, or dbx are updated. If an unsupported cryptographic algorithm is enrolled, `SetVariable()` still returns `EFI_SUCCESS`, but subsequent image verification will fail because the firmware cannot process the algorithm.

Proposal:

- Require firmware to validate that all algorithms in the updated PK/KEK/db/dbx entries are supported.
- Reject the update with an appropriate error if any enrolled algorithm is unsupported.
- Callers can query the EFI Crypto Indicator Table (ECIT) to determine which algorithms the firmware supports before enrollment.

UEFI Spec Change: In Chapter 32, three additions: - Section "Enrolling The Platform Key" — Add: "If the new Platform Key contains an unsupported algorithm or key size, the *SetVariable()* call shall return *EFI_UNSUPPORTED* and the new Platform Key shall not be enrolled." - Section "Enrolling Key Exchange Keys" — Add: "If the new Key Exchange Key contains an unsupported algorithm or key size, the *SetVariable()* call shall return *EFI_UNSUPPORTED* and the new Key Exchange Key shall not be enrolled." - Section "Signature Database Update" — Add: "If the new signature database entry contains an unsupported algorithm or key size, the *SetVariable()* call shall return *EFI_UNSUPPORTED* and the signature database shall not be updated."

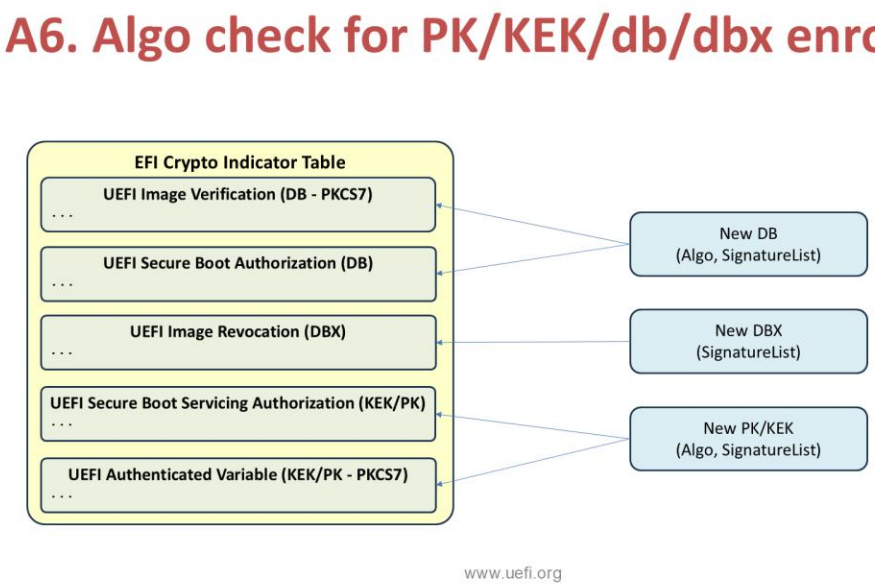

Figure 4: Algorithm Check for PK/KEK/db/dbx Enrollment

A7. Mandate One EFI_SIGNATURE_LIST per SetVariable() Call

Problem Statement: With the addition of algorithm checks (A6), callers need clear feedback on which specific algorithm is unsupported when a `SetVariable()` call is rejected.

Proposal:

- Mandate that each `SetVariable()` call to update the signature database provides signature list entries containing only one `EFI_SIGNATURE_LIST`.
- If the update is rejected, the caller can clearly identify why it failed.
- The caller may issue multiple `SetVariable()` calls to append entries with different `EFI_SIGNATURE_LIST` values, so the resulting database may contain multiple `EFI_SIGNATURE_LIST` entries.

UEFI Spec Change: In Chapter 32, two additions: - Section "Enrolling Key Exchange Keys" — Add: "Each *SetVariable()* call to update the Key Exchange Key database should provide signature list entries containing only one *EFI_SIGNATURE_LIST*. If the update contains more than one *EFI_SIGNATURE_LIST*, the *SetVariable()* call may return *EFI_UNSUPPORTED* and the Key Exchange Key database shall remain unchanged. The caller may issue multiple *SetVariable()* calls to append entries with different *EFI_SIGNATURE_LIST* values." - Section "Signature Database Update" — Add identical text for the image signature databases (db/dbx/dbt/dbr).

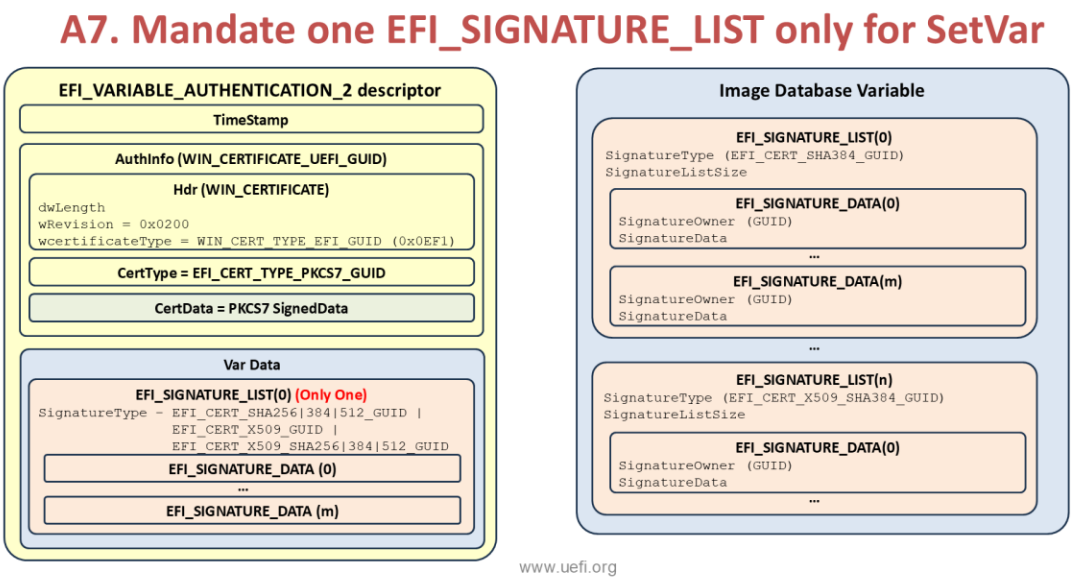

Figure 5: One EFI_SIGNATURE_LIST per SetVariable() Call

A8. KEK Self-Signed Append Operations

Problem Statement: Currently, the Key Exchange Key (KEK) can only be updated if signed by the Platform Key (PK). This creates a dependency on the platform owner for every KEK update, which may cause delays when an OS vendor needs to issue a new KEK.

Proposal:

- Allow a new KEK to be appended if it is signed by an existing KEK.
- This enables OS vendors to issue and append new KEKs independently, without requiring PK-signed updates for every change.
- The existing PK → KEK → DB/DBX trust hierarchy is maintained, with the addition of KEK → KEK append capability.

UEFI Spec Change: Changes in two chapters: - Chapter 32, Section "Enrolling Key Exchange Keys" — A new condition is added: "The platform is in user mode or deployed mode, and the *EFI_VARIABLE_APPEND_WRITE* attribute is set, and the provided variable data is signed with any current KEK_priv." Note clarifies that only append operations are permitted with KEK signing; full replacement or deletion still requires PK signing. - Chapter 8 (Runtime Services), `SetVariable()` step 5 — Updated: "If the variable is the global KEK variable, verify that the signature has been made with the current Platform Key, or if the *EFI_VARIABLE_APPEND_WRITE* attribute is set, verify that the signer's certificate chains to a certificate in the Key Exchange Key database." - Chapter 32, `EFI_CERT_EXTERNAL_MANAGEMENT_GUID` description — Updated to reflect that an append to KEK through *SetVariable()* may still succeed if signed by a valid KEKpriv.

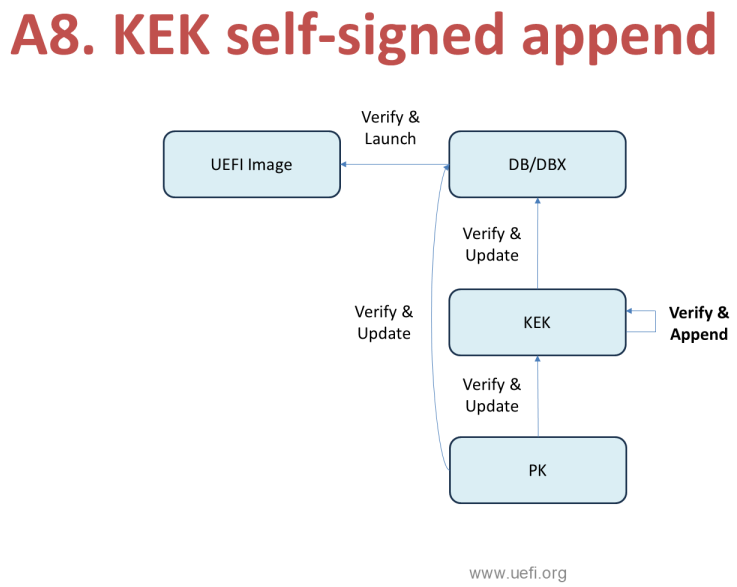

Figure 6: KEK Self-Signed Append Trust Hierarchy

B. Deprecations and Removals

B1. Deprecate "SignatureSupport" Variable

Problem Statement: The scope of the current `SignatureSupport` variable is unclear. The specification does not state whether it applies to UEFI Secure Boot only or to any crypto feature in the firmware.

Proposal: Deprecate the `SignatureSupport` variable and recommend using the ECIT table instead.

UEFI Spec Change: In Chapter 3 (Boot Manager), Section "Globally Defined Variables", a deprecation note is added: "**DEPRECATED: SignatureSupport Variable** — The *SignatureSupport* variable is deprecated and should no longer be used. This feature will be removed from future versions of the specification."

B2. Deprecate Private AuthVariable

Problem Statement: There is no specification-defined revocation mechanism for private authenticated variables. No mainstream OS has any intention of using them, and the current design allows an attacker to perform a pinning attack against a boot services variable that an OS depends on for security.

Proposal: Deprecate private authenticated variables.

UEFI Spec Change: Changes across multiple chapters: - Chapter 3 (Boot Manager) — Introduces a new "Secure Boot Policy Variables" section that explicitly enumerates PK, KEK, OsRecoveryOrder, OsRecovery####, db, dbx, dbt, and dbr as the defined Secure Boot Policy Variables with well-defined revocation mechanisms. - Chapter 8 (Runtime Services), Section `SetVariable()` — Adds deprecation note: "The use of the `EFI_VARIABLE_TIME_BASED_AUTHENTICATED_WRITE_ACCESS` attribute for variables other than Secure Boot Policy Variables is deprecated." - Chapter 32 (Secure Boot), Section "Signature Database Update" — Adds a cross-reference note to the Secure Boot Policy Variables definition.

B3. Remove EFI_VARIABLE_AUTHENTICATION_3 Descriptor

Problem Statement: No open-source implementation of `EFI_VARIABLE_AUTHENTICATION_3` exists, nor is there any interest in adoption from mainstream operating systems. This leads the specification to be misleading, as it informs the reader that they should enroll Secure Boot keys using `EFI_VARIABLE_AUTHENTICATION_3`.

Proposal: Remove `EFI_VARIABLE_AUTHENTICATION_3`.

UEFI Spec Change: Changes across multiple chapters: - Chapter 8 (Runtime Services) — Adds deprecation note: "All `EFI_VARIABLE_AUTHENTICATION_3` related definitions (`EFI_VARIABLE_ENHANCED_AUTHENTICATED_ACCESS`, `EFI_VARIABLE_AUTHENTICATION_3_CERT_ID_SHA256`, `EFI_VARIABLE_AUTHENTICATION_3_NONCE_TYPE`, `EFI_VARIABLE_AUTHENTICATION_3`, etc.) are deprecated and should no longer be used! They will be removed from future versions of the specification." - Chapters 3, 32 — All cross-references to `using-the-efi-variable-authentication-3-descriptor` are updated to point to `using-the-efi-variable-authentication-2-descriptor` instead.

B4. Remove Audit Mode and Deployed Mode from Secure Boot

Problem Statement: The specification describes an Audit Mode behavior that has never been widely adopted. Leaving it in the specification is confusing.

Proposal: Remove Audit Mode and Deployed Mode from the Secure Boot state machine, simplifying the mode transitions to Setup Mode and User Mode only.

UEFI Spec Change: Comprehensive removal across multiple chapters: - Chapter 3 (Boot Manager) — Remove `AuditMode` and `DeployedMode` from the Globally Defined Variables table. Simplify `SetupMode` description to remove references to AuditMode and DeployedMode. - Chapter 32 (Secure Boot) — Remove the "Transitioning to Audit Mode" and "Transitioning to Deployed Mode" sections entirely. Simplify trust relationship text: "While no Platform Key is enrolled, the platform is said to be operating in setup mode" and "After the

Platform Key is enrolled, the platform is operating in user mode." Remove all AuditMode-specific behavior from the Image Execution Information Table section. Update the Secure Boot Modes diagram.

C. Clarifications and Cleanup

C1. Clarify Ignoring Certificate Validity Check in Secure Boot

Problem Statement: The UEFI specification supports several time-based revocation mechanisms, all of which operate relative to times in structures. Nowhere does it address X.509 certificate *notBefore* and *notAfter* dates. Since the system clock in UEFI is not secure (and could be set to an arbitrary value either accidentally or on purpose), it cannot be relied upon to enforce absolute certificate expiry. The specification should state explicitly that certificate validity periods should not be compared against system time.

Proposal: Clarify that certificate validity period checks (e.g., notBefore/notAfter) should be ignored during Secure Boot signature verification.

UEFI Spec Change: In Chapter 32, Section "UEFI Image Validation — Overview", new text is added: "If a firmware supports the *EFI_CERT_X509_GUID* or *EFI_CERT_X509_SHAxxx_GUID* signature types, it should not compare the certificate validity period against system date and time. This is because the system date and time are insecure and may not be correctly set on some systems so doing validity comparisons could result in spurious and hard to diagnose image validation failures. The upshot is that any X509 certificate should always be treated as unexpired but it may be still revoked using a time based revocation."

C2. Clarify All Hash Algorithms for DBX

Problem Statement: The current UEFI specification only mentions `EFI_CERT_SHA256_GUID` for DBX, but all hash algorithms such as `EFI_CERT_SHA384_GUID` and `EFI_CERT_SHA512_GUID` should also be permitted.

Proposal: Clarify that all hash algorithms (`EFI_CERT_SHAxxx_GUID`) are applicable for the forbidden signature database (DBX).

UEFI Spec Change: In Chapter 32, Section "Authorization Process", the DBX matching criteria are generalized: - Item A: Changed from "Any entry with *SignatureListType* of *EFI_CERT_SHA256_GUID*" to "Any entry with *SignatureListType* of a hash GUID (such as *EFI_CERT_SHA256_GUID*, see Signature Database)" — making it algorithm-agnostic. - Item B: Changed from listing specific GUIDs (*EFI_CERT_X509_SHA256*, *EFI_CERT_X509_SHA384*, *EFI_CERT_X509_SHA512*) to "Any entry with *SignatureListType* of an X509 hash GUID (such as *EFI_CERT_X509_SHA256_GUID*, see Signature Database)" — making it extensible.

C3. Clarify Deprecation of SHA-1

Problem Statement: SHA-1 was deprecated long ago, but it is still listed in the UEFI specification. Its presence may cause confusion.

Proposal: Clarify that SHA-1 is deprecated for use in UEFI signature databases and Secure Boot operations.

UEFI Spec Change: In Chapter 32, Section "Creating Image Digests from Images", a new subsection "Supported Digests" is added: "SHA-1 was deprecated in 2011 and disallowed in digital signatures in 2013 by NIST so it is recommended that implementations not support it as a digest."

C4. Clean Up Language That Limits Crypto Agility

Problem Statement: The UEFI specification has several places where exact cryptographic requirements are specified. In these areas, the specification limits crypto agility by prescribing specific algorithms, key types, or encoding formats.

Proposal: Review and clean up specification language that inadvertently limits cryptographic agility, ensuring the specification can accommodate new algorithms as they are standardized.

UEFI Spec Change: Changes across three chapters: - Chapter 2 (Overview) — The entire "Cryptographic Algorithm Requirement" section is removed. This section had hardcoded specific algorithm mandates (SHA-256 OIDs, RSA-2048 key sizes, PKCS#1 v1.5 padding, TLS version recommendations) that limited crypto agility. - Chapter 28 (Network Protocols — TCP/IP and Configuration) — `EfiTlsConfigDataTypeCACertificate` description corrected from "PEM-encoded RSA or PKCS#8 private key" to "DER-encoded binary X.509 certificate or PEM-encoded X.509 certificate." - Chapter 37 (Secure Technologies) — `EFI_PKCS7_VERIFY_PROTOCOL.VerifyBuffer()` description changed from "a SHA256 or other hash" to "a hash" — removing the unnecessary SHA-256 callout to be algorithm-agnostic.

C5. Clarify One SignerInfo in AuthVariable Update

Problem Statement: The current UEFI specification is silent on the number of `SignerInfo` entries permitted in an authenticated variable update's PKCS#7 `SignedData`. This ambiguity may cause confusion regarding multiple `SignerInfo` usage.

Proposal:

- Mandate exactly one `SignerInfo` in authenticated variable updates.
- Do not support multiple `SignerInfo` entries.
- Composite algorithms (e.g., ML-DSA + ECDSA composite defined by IETF) are supported as a single algorithm with a single `SignerInfo`.
- The PKCS#7 `SignedData` structure for authenticated variable updates:
    - `version` = 1
    - `digestAlgorithm` = SHA256 | SHA384 | SHA512
    - `Certificates` = one signer's certificate
    - `SignerInfos` = one `SignerInfo`
    - `digestEncryptionAlgorithm` = RSA | ECDSA | ML-DSA

UEFI Spec Change: In Chapter 8 (Runtime Services), two additions to the PKCS#7 `SignedData` construction steps: - Section `EFI_VARIABLE_AUTHENTICATION_2` descriptor — Add: "Only one SignerInfo shall be present."

4. UEFI CA PQC Plan

The UEFI Certificate Authority (CA) will be updated to support PQC:

- Follow current behavior with 2011/2023 UEFI CA.
- Support multiple signatures.
- Continue using SHA256 revocation.

5. Summary

The UEFI specification PQC updates represent a comprehensive effort to prepare the UEFI ecosystem for the post-quantum era. The changes span three categories:

Category	Updates
A. New Features	Multiple signature verification, TBSCertificate hash in DB, Certificate removal in DBX, EFI Crypto Indicator Table, algorithm enrollment checks, one EFI_SIGNATURE_LIST per SetVariable(), KEK self-signed append
B. Deprecations/Removals	SignatureSupport variable, Private AuthVariable, EFI_VARIABLE_AUTHENTICATION_3, Audit/Deployed Modes
C. Clarifications	Ignore cert validity in Secure Boot, all hash algorithms for DBX, SHA-1 deprecation, crypto agility language cleanup, one SignerInfo

The guiding philosophy is to make the UEFI infrastructure crypto-agile — ready for PQC without mandating specific algorithms — while reusing industry-standard PQC definitions from NIST and IETF.

6. References

Standards and Guidance

Source	Document	Link
NIST	FIPS 203 — ML-KEM	https://csrc.nist.gov/pubs/fips/203/final
NIST	FIPS 204 — ML-DSA	https://csrc.nist.gov/pubs/fips/204/final
NIST	FIPS 205 — SLH-DSA	https://csrc.nist.gov/pubs/fips/205/final
NIST	SP 800-227 — KEM Recommendation	https://csrc.nist.gov/pubs/sp/800/227/final
NIST	IR 8547 — PQC Transition (draft)	https://csrc.nist.gov/pubs/ir/8547/ipd
NIST	PQC FAQ	https://csrc.nist.gov/projects/post-quantum-cryptography/faqs
NSA	CNSA 2.0 Algorithms	https://media.defense.gov/2025/May/30/2003728741/-1/-1/0/CSA_CNSA_2.0_ALGORITHMS.PDF
NSA	CNSA 2.0 FAQ	https://media.defense.gov/2022/Sep/07/2003071836/-1/-1/0/CSI_CNSA_2.0_FAQ_.PDF
IETF	ML-DSA Certificates (RFC 9881)	https://datatracker.ietf.org/doc/rfc9881/
IETF	ML-DSA CMS (RFC 9882)	https://datatracker.ietf.org/doc/rfc9882/
IETF	SLH-DSA Certificates (RFC 9909)	https://datatracker.ietf.org/doc/rfc9909/
IETF	SLH-DSA CMS (RFC 9814)	https://datatracker.ietf.org/doc/rfc9814/
IETF	Composite	https://datatracker.ietf.org/doc/draft-ietf-lamps-pq-composite-sigs/

	ML-DSA Signatures	
IETF	Hybrid TLS	https://datatracker.ietf.org/doc/draft-ietf-tls-hybrid-design
EU	PQC Transition Roadmap	https://digital-strategy.ec.europa.eu/en/library/coordinated-implementation-roadmap-transition-post-quantum-cryptography
UK NCSC	PQC Migration Timelines	https://www.ncsc.gov.uk/guidance/pqc-migration-timelines
UK NCSC	Next Steps for PQC	https://www.ncsc.gov.uk/whitepaper/next-steps-preparing-for-post-quantum-cryptography
Germany BSI	PQC Joint Statement	https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Crypto/PQC-joint-statement.pdf
France ANSSI	PQC Position Paper	https://messervices.cyber.gouv.fr/guides/en-follow-position-paper-post-quantum-cryptography
PQShield	Secure Boot Considerations with PQC	https://pqshield.com/secure-boot-considerations-with-pqc/
UEFI Forum	PQC Impact Webinar	https://uefi.org/sites/default/files/resources/Post%20Quantum%20Webinar.pdf