

Technical White Paper

Demystifying Post-Quantum Cryptography for Embedded Systems



Walter Schnoor

Abstract

What is PQC?

Post-quantum cryptography (PQC) is a category of cryptographic primitives designed to protect against an adversary in possession of both classical computing as well as cryptographically relevant quantum computing capability.

Table of Contents

1 Introduction.....	2
2 Security Goals in Embedded Systems.....	2
3 Cryptographic Primitives.....	2
4 The Potential Effects of CRQCs.....	2
5 New Algorithms and Standards for the Future.....	3
6 CNSA Compliance.....	3
7 Symmetric Encryption and Hash Functions.....	4
8 PQC Digital Signatures: ML-DSA.....	4
9 Is Hardware Acceleration Necessary?.....	5
10 Is Pre-Hash Necessary?.....	6
11 Other NIST-Selected PQC Digital Signature Schemes.....	6
12 Stateful Digital Signature Schemes.....	7
13 What About Key Agreement?.....	7
14 Future-Proofing Embedded Systems.....	8
15 References.....	9

Trademarks

Arm® and Cortex® are registered trademarks of Arm Limited.
All trademarks are the property of their respective owners.

1 Introduction

While post quantum cryptography can sound mysterious, future-proofing your embedded system against future quantum-capable adversaries does not have to be. This is because while PQC changes the underlying algorithms used to achieve security, it does not change your high-level security goals. In other words, the "what" is still the same- it's the "how" that changes. This white paper explains how PQC fits in with the common security goals in an embedded system, the industry standards relating to PQC, and the implications on resource-constrained embedded systems using microcontrollers- what it means for latency, memory, and communication time.

2 Security Goals in Embedded Systems

To understand the implications of PQC on an embedded system, it's best to start from the security goals themselves. Many embedded systems share a common set of security goals, including:

- Secure boot, which makes sure that the software or firmware running on a processor is authentic – in other words, that the manufacturer of the system authorized the software or firmware.
- Secure update, which keeps new software and firmware updates confidential in the interest of protecting intellectual property.
- Attestation, which enables a device or system to attest to its knowledge of a secret, to prove to a cloud server or another module in a system that it is genuine and can be trusted.
- Secure communication, which preserves the confidentiality, integrity and authenticity of messages between devices in an embedded system or between an embedded system and a remote server.

Deploying cryptographic primitives in a cryptographic system achieves these security goals. For example, a microcontroller in an electronic control unit can implement secure boot by including immutable root-of-trust firmware that uses a digital signature algorithm to authenticate application software before allowing it to run on the CPU, thus preventing the execution of unauthorized code on the CPU. If the underlying primitives become compromised in the future, then security goals such as secure boot can, by extension, become compromised.

3 Cryptographic Primitives

Cryptographic primitives are the core building blocks in cryptography required to build a cryptographic system. One example of a cryptographic primitive is a symmetric block cipher, which uses a single secret key to encrypt plaintext into ciphertext, and to decrypt ciphertext back into plaintext. Another example is a digital signature algorithm, which can verify the authenticity of a message using two keys: a secret key for signing messages as authentic, and a public key to verify that a message was signed by the matching secret key. These primitives derive their security from what is known as hardness assumptions.

One of the most popular digital signature algorithms in use today is the elliptic curve digital signature algorithm (ECDSA), which bases its security on the hardness of the elliptic curve discrete logarithm problem (ECDLP). Currently, there is no known algorithm for solving ECDLP efficiently using the computers of today. Thus, ECDLP is considered "hard" enough to rely on for security. Quantum computers, however, are another story.

4 The Potential Effects of CRQCs

Classical computing refers to today's computers, which run defined operations as a series of discrete instructions. It is possible to parallelize these instructions in some situations, but the computing is limited to sequential operation or parallel sets of sequential operations.

Quantum computers, on the other hand, can take advantage of data bits that can be both "0" and "1" at the same time, allowing them to search a solution space simultaneously in a way that the computers of today cannot [1].

A cryptographically relevant quantum computer (CRQC) would theoretically be able to solve the ECDLP through the use of known algorithms. If such a quantum computer were to exist, it would break the hardness assumptions that make today's popular cryptographic primitives secure, such as ECDSA. An adversary could compromise the cryptographic primitive, thus compromising the security goal supported by the cryptographic primitive. A primary goal in attacking ECDSA is to recover the private key which is used to sign new messages as authentic, such that an adversary can then use the private key to sign fraudulent messages, producing valid signatures. This requires finding the private key value k given the public key P and public generator point G ,

where $P = kG$ on an elliptic curve. With a classical computer, recovering k given the public values P and G is hard (known methods run in exponential time). With a CRQC, it can be possible to run Shor's algorithm: a quantum algorithm which reduces the attack from exponential time to polynomial time, making it practical to execute. At the time of this writing, quantum computers that are large and stable enough to execute algorithms – such as Shor's algorithm – that can compromise popular implementations of ECDSA (such as ECDSA using the National Institute of Standards and Technology [NIST] P-256 curve) do not yet exist, but they may in the future.

While no one knows when a powerful enough quantum computer will exist that can break the classical cryptographic primitives of today [1], many industries are beginning to secure their products against future threats for these reasons:

- Building trusted new cryptography takes time, as does integrating it into products.
- Many embedded systems have extended lifetimes of 10 or more years in the field. Addressing the potential invention of CRQCs now reduces the risk of lost security in the future.
- Many markets are driving regulation, especially the enterprise and data center market.

As a result, we are seeing more embedded system designers moving to PQC cryptographic primitives to achieve security goals.

5 New Algorithms and Standards for the Future

The security goals for most embedded systems are independent of the cryptographic primitives (classical or quantum) used to achieve them. The motivation to adopt PQC is not to change the security goals for an embedded system; it's to change the underlying arithmetic used to achieve the security goal such that the hardness problems are protected against attackers with CRQC capability.

6 CNSA Compliance

In the United States, products used in national security systems must comply with the National Security Agency's (NSA) commercial national security algorithm (CNSA) suite. In 2022, the NSA updated the CNSA suite from version 1.0 to version 2.0; one of the primary motivations for CNSA 2.0 was the incorporation of PQC. New deployments of the CNSA suite must comply with version 2.0 by Jan. 1, 2027, while already deployed systems must comply or be replaced with CNSA 2.0-compliant systems by Dec. 31, 2030.

[Table 6-1](#) lists the cryptographic primitives approved for CNSA 1.0 versus the required cryptographic primitives approved for CNSA 2.0.

Table 6-1. Comparing Approved Cryptographic Primitives for CNSA 1.0 and CNSA 2.0

Application usage	Cryptographic primitive	CNSA 1.0 approved	CNSA 2.0 approved [2]
General-purpose cryptography	Symmetric encryption	Federal Information Processing Standard (FIPS) publication (PUB) 197 (Advanced Encryption Standard [AES]-256 only)	
	Secure hash	FIPS PUB 180-4 (secure hash algorithm [SHA]-384 only)	FIPS PUB 180-4 (SHA-384 or SHA-512)
	Digital signature	FIPS PUB 186-4 (ECDSA P-384 or Rivest, Shamir and Adleman 3,072-bit modulus)	FIPS PUB 204 (Module Lattice Based-Digital Signature Standard [ML-DSA]-87 only)
	Key agreement	Elliptic Curve Diffie-Hellman P-384 or Finite Field Diffie-Hellman (3,072-bit modulus)	FIPS PUB 203 (Module Lattice Based-Key-Encapsulation Mechanism [ML-KEM]-1,024 only)
Firmware or software signing only	Digital signature	–	NIST special publication (SP) 800-208 (Leighton-Micali Signature [LMS] or eXtended Merkle Signature Scheme [XMSS] [1])
Hardware integrity checking only	Secure hash	–	FIPS PUB 202 (SHA-3-384 or SHA-3-512)

7 Symmetric Encryption and Hash Functions

CNSA 2.0 does not substantially change the minimum requirements for symmetric encryption primitives and secure hash functions. For example, CNSA 2.0 preserves the same algorithm and security level for symmetric encryption (AES-256) as was used in CNSA 1.0. This is because a CRQC does not pose a significant threat today against AES-256. While there are quantum algorithms that can reduce the security of AES, the key size selection in CNSA 1.0 and 2.0 (256 bits) is high enough that the primitive remains secure. Take Grover's algorithm as an example. Grover's algorithm is a quantum algorithm which can reduce the brute-force attack time on AES from 2^k down to $2^{k/2}$ where k is the key size, in bits. In the best case, the security strength of AES-256 would reduce from 2^{256} to 2^{128} when using Grover's algorithm, which is still considered secure. By contrast, designs using AES-128 can see a reduction in security strength from 2^{128} to 2^{64} , which can be vulnerable. As a result, in applications where AES-128 is used today, it is recommended to switch to AES-256, thus nullifying the benefit of a quantum computer.

Some of the most significant changes in CNSA 2.0 relate to digital signatures and key agreement.

8 PQC Digital Signatures: ML-DSA

For general-purpose digital signatures, including firmware and software update signatures, CNSA 2.0 requires the use of ML-DSA. ML-DSA relies on the module learning with errors hardness problem for its security, and is believed to be secure against adversaries with both classical and quantum computing capability. The design of ML-DSA is based on the Crystals-Dilithium signature scheme submission to NIST, which NIST selected for standardization as FIPS 204 after three rounds of analysis and evaluation [4].

In embedded applications using ECDSA for digital signatures (including secure boot, attestation and other signature use cases), ML-DSA provides a post-quantum secure alternative with similar programming interfaces. ML-DSA is also scalable; it supports three different security levels (ML-DSA-44, ML-DSA-65 and ML-DSA-87), each with trade-offs between key and signature size, computing latency, computing static random access memory (SRAM) space, and security. CNSA 2.0 mandates the highest security level: ML-DSA-87.

For embedded system developers, ML-DSA differs from ECDSA significantly in these ways:

- ML-DSA has much larger key and signature sizes than ECDSA. The signature size is more than 48 times larger for ML-DSA-87 (4,627 bytes) than it is for ECDSA P-384 (96 bytes), requiring more provisioning in flash and SRAM and longer transmission times on the wire when performing attestation functions or installing new software and firmware updates.
- ML-DSA verification tends to be faster than ECDSA verification when comparing CPU-based implementations, reducing authentication time. The ML-DSA-87 signature verification operation, for instance, can execute in one-tenth the time of ECDSA over the NIST P-384 curve, improving boot times significantly for microcontrollers.
- ML-DSA implementations typically require more SRAM for performing the necessary computations when compared to ECDSA, with different trade-offs between working memory and execution latency.

Table 8-1 compares classical computing signatures (ECDSA P-384) to post-quantum computing signatures (ML-DSA) for each security level of ML-DSA.

Table 8-1. Comparison of ECDSA Parameter Sizes and Performance with ML-DSA

Parameter	ECDSA	ML-DSA-44	ML-DSA-65	ML-DSA-87
Security level	P-384	Category 2	Category 3	Category 5
Secret key (bytes)	48	2,570	4,032	4,896
Public key (bytes)	96	1,312	1,952	2,592
Signature (bytes)	96	2,420	3,309	4,627
Signature wire time (seconds) ⁽¹⁾	0.009	0.22	0.30	0.42
Keygen time (seconds) ⁽²⁾	1.12	0.084	0.14	0.23
Signature time (seconds) ⁽²⁾	1.23	≈0.4	≈0.8	≈1.6
Verification time (seconds) ⁽²⁾	2.26	0.10	0.16	0.26
SRAM requirement for verification (kilobytes) ⁽²⁾	2.2	21.9	29.1	37.6

(1) Transmission time for a signature at a 100kHz rate at 9 bits per byte

(2) Test environment is a Texas Instruments (TI) 80MHz Arm® Cortex®-M0+ microcontroller running the TI MSP Crypto Library

9 Is Hardware Acceleration Necessary?

Developers building embedded systems that use elliptic curve cryptography may use microcontrollers with acceleration for the elliptic curve computation, if the application is latency sensitive. This can, however, lead to the selection of a microcontroller that is more expensive than what the application actually requires.

For example, in a challenge-response application, computing an ECDSA P-384 signature for a challenge-response scheme can take more than 2s on an 80MHz Arm Cortex-M0+ processor, which is far longer than the 6ms that it takes to transmit the resulting signature on the wire at 100Kbps (a typical rate, assuming communication methods such as standard I²C or 115200 serial universal asynchronous receiver transmitter).

When moving to PQC, the larger digital signatures and shorter computation times change this dynamic significantly. For example, the wire time required to transmit a signature increases by 48 times, from approximately 9ms (for ECDSA P-384) to approximately 420ms (for ML-DSA-87). Signature verification computation time reduces from approximately 2,200ms (for ECDSA P-384) to approximately 260ms, which is nearly one-half the time it takes to transmit the signature on the wire. As a result, applications such as attestation tend to be more limited by communication time than by computation time, making CPU-based implementations economical. [Figure 9-1](#) compares the verify performance for each security level on Arm Cortex-M0+ and Arm Cortex-M33 target MCUs.

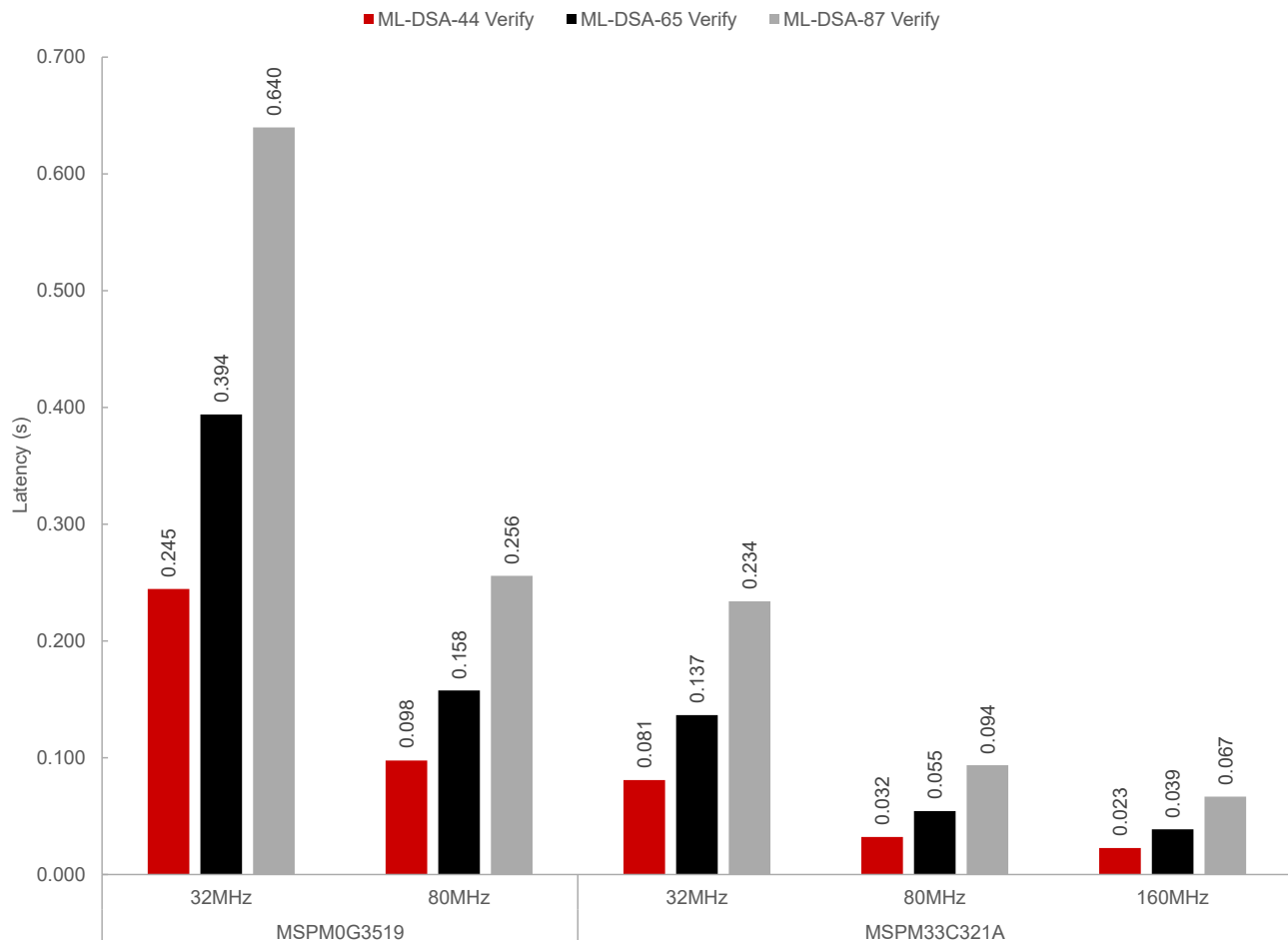


Figure 9-1. ML-DSA Verify Performance for TI MSPM0, MSPM33 Microcontrollers

10 Is Pre-Hash Necessary?

In secure boot implementations, it is common to first hash the application software and firmware image to create a fixed-size message representative using a secure hash function such as SHA-256 or SHA-384, which is then signed and verified through ECDSA. ML-DSA eliminates this "pre-hash" step. In fact, the designers of ML-DSA prefer that users of the scheme pass the entire message content for a digital signature directly into the signature algorithm itself.

In FIPS PUB 204, this process is called "pure" mode. FIPS PUB 204 also defines a pre-hash mode known as HashML-DSA, which includes information about the hash function used in the computation to generate the signature (thus binding the hash function to the signature).

Beyond FIPS 204, it is also possible to employ a third option: pre-hashing the message but signing the hash using ML-DSA in pure mode as if the message was passed directly to ML-DSA, rather than using the pre-hash HashML-DSA mode of FIPS 204. This leaves out the binding of the hash function to the signature, but it is more comparable to how ECDSA is used with SHA today.

ML-DSA uses the Keccak function internally when generating and verifying digital signatures. Keccak is the core function used inside of the SHA-3 function and the SHA and Keccak (Shake) extendable-output function. There are many ways that ML-DSA uses Keccak, including secure digest of the message and deterministic expansion of the public key during verification.

This matters because on resource-constrained systems, the inclusion of ML-DSA means that Keccak is also included, providing a secure hash function within the same code footprint. As a result, using the pure ML-DSA mode of operation means that you do not need to give up additional nonvolatile code memory to store an additional hash function such as SHA-384 or SHA-512. Additionally, Keccak is available for exposing SHA-3 as a general-purpose hash function for other purposes in an application.

At this time, CNSA 2.0-compliant systems can pre-hash a message using an approved secure hash function (SHA-384 or SHA-512) but cannot use ML-DSA in its pre-hash mode (HashML-DSA) as specified in FIPS PUB 204; rather, FIPS PUB 204 pure mode is the default.

11 Other NIST-Selected PQC Digital Signature Schemes

In addition to ML-DSA, NIST also selected two post-quantum computing digital signature schemes for standardization:

- The Stateless Hash Based-Digital Signature Standard (SLH-DSA) based on the Sphincs+ signature scheme, standardized as FIPS PUB 205.
- The fast Fourier transform over Nth degree truncated polynomial ring unit lattice based-digital signature algorithm (FN-DSA), or Falcon, which is in the process of being standardized as FIPS PUB 206.

SLH-DSA is a hashed-based signature scheme. Like ML-DSA, it is stateless. Unlike ML-DSA, the higher security levels have very large signature sizes (in the tens of kilobytes), which limit the applicability of SLH-DSA in embedded applications, especially microcontroller-based applications. At this time, CNSA 2.0 does not permit SLH-DSA-based signatures, even though NIST standardized SLH-DSA.

FN-DSA promises smaller signatures than ML-DSA and SLH-DSA, at the expense of complexity. FN-DSA depends on floating-point arithmetic, which is notoriously difficult to implement in constant time across processor architectures. Additionally, not all embedded architectures contain a floating-point unit, giving back the performance benefit. Currently, CNSA 2.0 does not permit FN-DSA [3], and NIST has not yet standardized FN-DSA.

For these reasons, the industry is largely moving toward ML-DSA as the de facto digital signature scheme, as it balances performance, size and security well.

12 Stateful Digital Signature Schemes

ML-DSA and the schemes discussed are all stateless schemes. There is no requirement to keep the state during signing for a given private key, which is a sought-after quality in embedded applications, especially in attestation use cases where there is frequent challenge-response signing occurring during normal operation in resource-constrained environments.

Before the stateless schemes were standardized, several post-quantum secure stateful signature schemes provided options for specific use cases such as the signing of software and firmware images (secure boot), including LMS and XMSS as standardized in NIST SP 800-208.

The smaller public keys offered by these schemes (smaller than ML-DSA) are beneficial for embedded systems implementing secure boot because they require less embedded nonvolatile memory (usually flash memory) when authenticating new firmware. The signing operation does require tracking of the state for a given private signing key to ensure the secrecy of the private key. This may be acceptable for firmware and software updates, as it is possible to maintain the state for a signing hardware security module in a controlled IT environment, where updates to firmware and software images are infrequent. It is often not acceptable in an embedded environment, however.

As a result, LMS and XMSS are permissible in CNSA 2.0-compliant systems, but only for use in firmware and software digital signatures; they cannot be used for other use cases such as attestation. By contrast, it is possible to use ML-DSA for all digital signature use cases, including secure boot and attestation.

13 What About Key Agreement?

Key agreement is a necessary step to establish a shared secret between two entities over an unsecure channel. Like digital signatures, key agreement algorithms relying on today's popular primitives such as finite-field Diffie-Hellman (FFDH) or elliptic curve Diffie-Hellman (ECDH) are also vulnerable to quantum based attacks using Shor's algorithm. To address this, NIST has standardized ML-KEM (the module lattice key encapsulation mechanism) in FIPS 203. ML-KEM is a post quantum secure key encapsulation mechanism (KEM) designed to be used in scenarios where FFDH or ECDH are used today.

Unlike Diffie-Hellman style designs that derive a shared secret from randomness on both sides of the handshake, a KEM has a key generation step, an encapsulation step, and a decapsulation step. In a client-server model, the process for key agreement with ML-KEM is as follows:

1. A key pair is first generated by the server. The encapsulation (or "public") key is distributed to clients. The decapsulation (or "private") key is held back by the server.
2. When a client wants to establish a secure channel with the server and agree on a symmetric key, it performs an ML-KEM encapsulation operation based on a random seed, out of which it obtains a ciphertext and the shared secret. It then sends the ciphertext to the server.
3. The server receives the ciphertext and performs a decapsulation operation, giving the same shared secret as the client.
4. The server and client may now communicate using the shared secret key.

ML-KEM can be performant in software, with encapsulation times as low as 7ms on a 160MHz Cortex-M33 MCU. [Figure 13-1](#) compares the encapsulation performance for each security level on Arm Cortex-M0+ and Arm Cortex-M33 target MCUs.

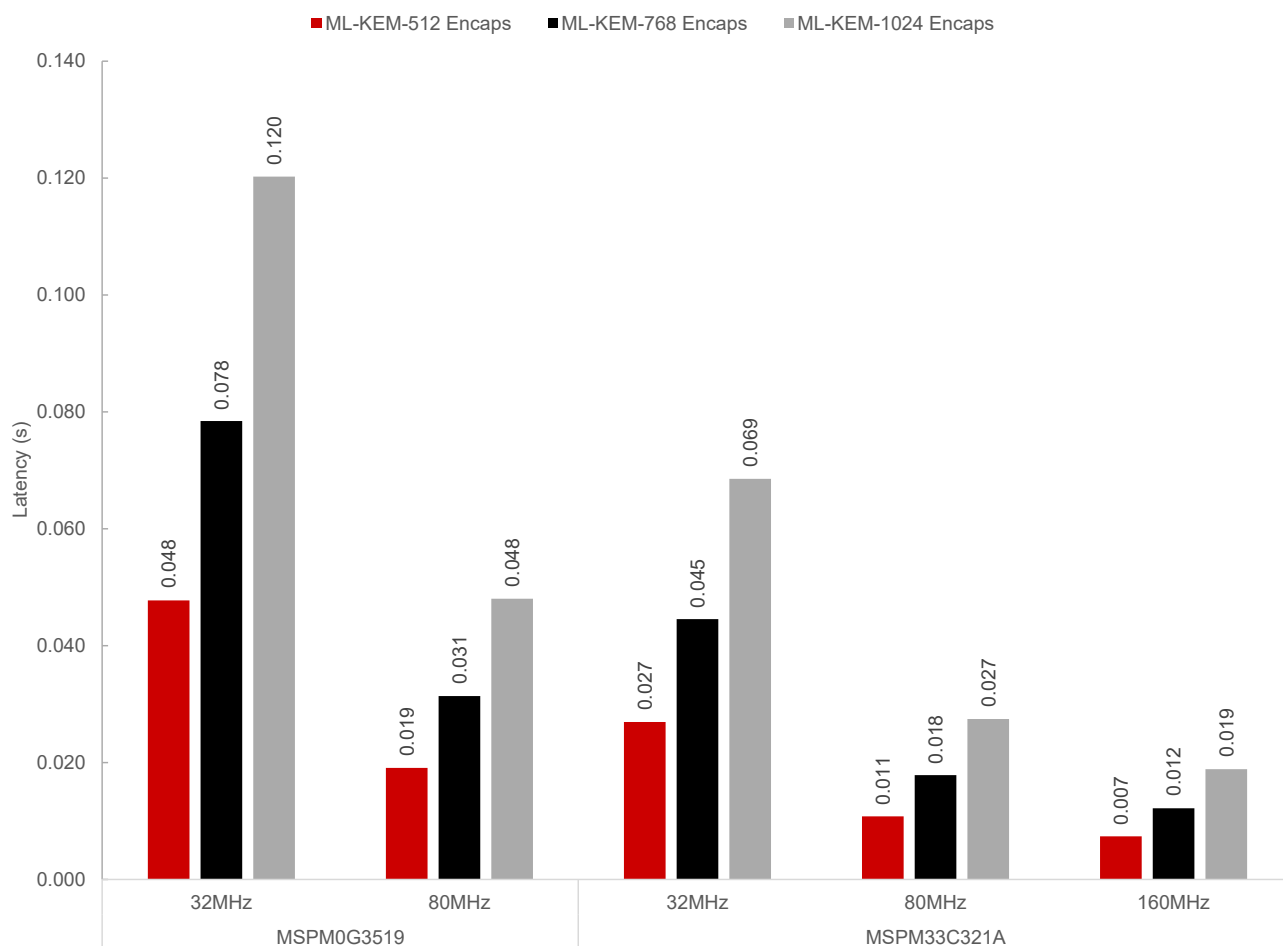


Figure 13-1. ML-KEM Encapsulation Performance for TI MSPM0, MSPM33 Microcontrollers

ML-KEM has other desirable security properties beyond being designed to be quantum secure. Notably, it supports IND-CCA2 (indistinguishability under a chosen ciphertext attack). This means that even if an attacker can manipulate the ciphertext in transit from the client to the server (between encapsulation and decapsulation), the attacker would not gain an advantage or additional knowledge because the output of decapsulation is always indistinguishable from random. Additionally, the output of ML-KEM is usable directly as a symmetric encryption key, bringing simplicity to embedded applications and potentially removing the need for extra key derivation steps.

14 Future-Proofing Embedded Systems

Designers need to consider the possibility that an attacker can, in the future, possess a quantum computer with enough capability to attack classical cryptographic primitives used to provide basic security goals, including secure boot, attestation and secure communication. Through the NIST and NSA, the United States has a framework for selecting cryptographic primitives believed to be secure even against an attacker with such capabilities.

Challenges in embedded systems include small nonvolatile and working memory, slower communication interfaces, and a limited ability to store the state. TI provides support for post-quantum secure cryptographic primitives on ultra-low-power and low-cost microcontrollers that enable CNSA 2.0 compliance and balance performance and memory, including:

- Digital signatures: full support for ML-DSA-44, ML-DSA-65 and ML-DSA-87 in pre-hash and pure forms.
- Key encapsulation: full support for ML-KEM-512, ML-KEM-768, and ML-KEM-1024.
- Symmetric encryption: support for AES-256 and associated modes of operation, including counter with cipher block chaining-message authentication code and Galois counter mode.

- Secure hash functions: full support for SHA-2 and the SHA-3 and SHAKE suites.

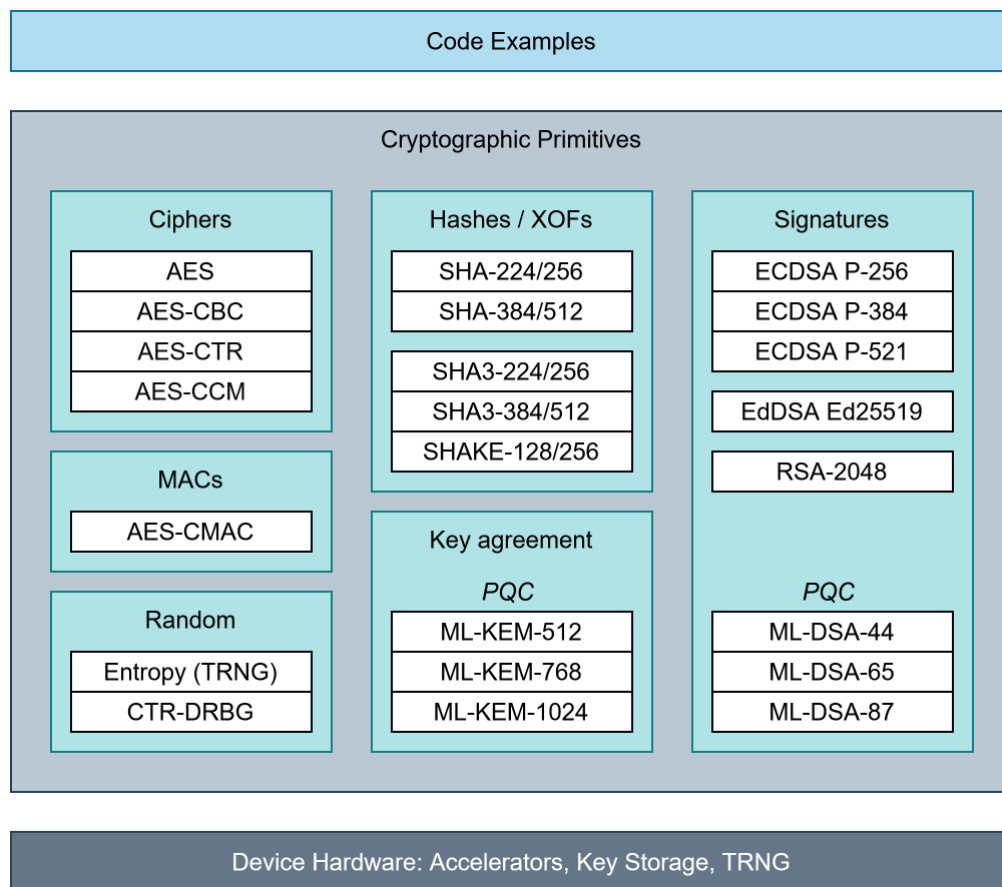


Figure 14-1. MSP-CRYPTO-LIB Features

Using the TI MSP Crypto Library on MSPM0 Arm Cortex-M0+ and MSPM33 Arm Cortex-M33-based microcontrollers enables CNSA 2.0-compliant ML-DSA-87 digital signature verification for secure boot use cases in less than 260ms (Cortex-M0+) or 70ms (Cortex-M33), providing fast boot times even in resource- and cost-constrained environments, while preparing for a future where quantum computing is a reality.

15 References

- NIST. Aug. 13, 2024. ["What is Post-Quantum Cryptography?"](#) Accessed Feb. 3, 2026.
- NSA. September 2022. ["Announcing the Commercial National Security Algorithm Suite 2.0."](#) Accessed Feb. 3, 2026.
- NSA. December 2024. ["The Commercial National Security Algorithm Suite 2.0 and Quantum Computing FAQ."](#) Accessed Feb. 3, 2026.
- NIST FIPS 204. Aug. 13, 2024. ["Module-Lattice-Based Digital Signature Standard."](#) Accessed June 6, 2026.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you fully indemnify TI and its representatives against any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI's products are provided subject to [TI's Terms of Sale](#), [TI's General Quality Guidelines](#), or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI's provision of these resources does not expand or otherwise alter TI's applicable warranties or warranty disclaimers for TI products. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

TI objects to and rejects any additional or different terms you may propose.

Copyright © 2026, Texas Instruments Incorporated

Last updated 10/2025