PQUIP
Internet-Draft
Intended status: Informational
Expires: 6 January 2026

T. Reddy
Nokia
D. Wing
Cloud Software Group
B. Salter
UK National Cyber Security Centre
K. Kwiatkowski
PQShield

5 July 2025

Adapting Constrained Devices for Post-Quantum Cryptography
draft-ietf-pquip-pqc-hsm-constrained-01

Abstract

This document offers guidance on incorporating Post-Quantum Cryptography (PQC) into resource-constrained devices, including IoT devices and lightweight Hardware Security Modules (HSMs), which operate under tight limitations on compute power, memory, storage, and energy. It highlights how the Root of Trust acts as the foundation for secure operations, enabling features such as seed-based key generation to reduce the need for persistent storage, efficient approaches to managing ephemeral keys, and methods for offloading cryptographic tasks in low-resource environments. Additionally, it examines how PQC affects firmware update mechanisms in such constrained systems.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-pquip-pqc-hsm/.

Discussion of this document takes place on the pquip Working Group mailing list (mailto:pqc@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/pqc/. Subscribe at https://www.ietf.org/mailman/listinfo/pqc/.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

This Internet-Draft will expire on 6 January 2026.

Copyright Notice

Table of Contents

1. Introduction 3
2. Key Management in Constrained Devices for PQC 3
  2.1. Seed Management 4
    2.1.1. Seed Storage 4
    2.1.2. Efficient Key Derivation 6
    2.1.3. Exporting Seeds and Private Keys 6
3. Ephemeral Key Management 7
4. Optimizing Performance in Constrained Devices for PQC Signature Algorithms 8
5. Additional Considerations for PQC Use in Constrained Devices 10
  5.1. Key Sizes of Post-Quantum Algorithms 10
6. Post-quantum Firmware Upgrades for Constrained Devices . . . 11
  6.1. Post-quantum Firmware Authentication 12
  6.2. Hybrid signature approaches 13
7. Impact of PQC Authentication on Constrained Devices 13
8. Security Considerations 15
  8.1. Side Channel Protection 15
Acknowledgements 15
References 15

Normative References 15
Informative References 16
Authors' Addresses 18

1. Introduction

The transition to post-quantum cryptography introduces significant challenges for resource-constrained devices such as IoT devices, lightweight HSMs, secure elements (e.g., SIMs), and Trusted Execution Environments (TEEs). These devices often operate with limited memory, non-volatile storage, processing power, and battery life, making the adoption of PQC algorithms which typically involve larger key sizes and are more computationally intensive than traditional algorithms particularly challenging. The increased key sizes and computational demands of PQC require careful consideration to ensure secure and efficient key management within these constrained environments.

This document provides industry guidance and best practices for integrating PQC algorithms into constrained devices. It explores key storage strategies, ephemeral key management, and performance optimizations specific to resource-limited environments. One approach to mitigating storage constraints is seed-based key generation, where only a small seed is stored instead of the full private key, as supported by PQC schemes like ML-DSA and SLH-DSA. However, this technique increases computational overhead due to the need to derive full private keys on demand, a classic computation-versus-storage tradeoff. The document also discusses considerations for ephemeral key generation in protocols like TLS and IPsec, along with techniques to optimize PQC signature operations to enhance performance within constrained crytographic modules.

This document focuses on post-quantum cryptography in constrained devices, specifically on the three algorithms finalized by NIST: ML-DSA, ML-KEM, and SLH-DSA and on stateful hash-based signatures in the context of firmware signing. Future revisions may expand the scope to include additional PQC algorithms.

2. Key Management in Constrained Devices for PQC

The embedded cryptographic components used in constrained devices are designed to securely manage cryptographic keys, often under strict limitations in memory, storage capacity, and computational resources. These limitations are further exhausted by the increased key sizes and computational demands of PQC algorithms.

One mitigation of storage limitations is to store only the seed rather than the full expanded private key, as the seed is far smaller and can derive the expanded private key as necessary.

2.1. Seed Management

The seed generated during the PQC key generation function is highly sensitive, as it will be used to compute the private key or decapsulation key. Consequently, seeds must be treated with the same level of security as private keys.

To comply with [ML-KEM], [ML-DSA], [SLH-DSA] and [REC-KEM] guidelines:

2.1.1. Seed Storage

Some PQ key exchange mechanisms use a seed to generate their private keys (e.g., ML-KEM, McEliece, and HQC), and those seeds are smaller than private keys, saving storage space. Some implementations may choose to retain the (small) seed rather than the (larger) private key. As the private key is necessary for cryptographic operations, it can be derived from the seed when needed or retained in a cache within the security module.

Seeds must be securely stored within a cryptographic module of the device whether hardware or software-based to protect against unauthorized access. Since the seed can derive the private key, it must be safeguarded with the same level of protection as a private key. For example, according to [ML-DSA] Section 3.6.3, the seed 両 generated during ML-DSA.KeyGen can be stored for later use with ML-DSA.KeyGen_internal.

The choice between storing a seed or an expanded private key involves trade-offs between storage efficiency and performance. Some constrained cryptographic modules may store only the seed and derive the expanded private key on demand, whereas others may prefer storing the full expanded key to reduce computational overhead during key usage.

While vulnerabilities like the "Unbindable Kemmy Schmidt" attack [BIND] demonstrate the risks of manipulating expanded private keys in environments lacking hardware-backed protections, these attacks generally assume an adversary has some level of control over the expanded key format. However, in a hardware-backed protcted environment, where private keys are typically protected from such manipulation, the primary motivation for storing the seed rather than the expanded key is not directly tied to mitigating "Kemmy" attacks.

The ML-DSA and ML-KEM private key formats, as specified in [I-D.ietf-lamps-dilithium-certificates] and [I-D.ietf-lamps-kyber-certificates], represent the private key using a seed from which the expanded private key is derived. While these formats rely on the seed for key generation, an constrained cryptographic module may choose to store the expanded private key to avoid the additional computation required for running KeyGen.

This choice between storing the seed or the expanded private key has direct implications on performance, as key derivation incurs additional computation. The impact of this overhead varies depending on the algorithm. For instance, ML-DSA key generation, which primarily involves polynomial operations using the Number Theoretic Transform (NTT) and hashing, is computationally efficient compared to other post-quantum schemes. In contrast, SLH-DSA key generation requires constructing a Merkle tree and multiple calls to Winternitz One-Time Signature (WOTS+) key generation, making it significantly slower due to the recursive hash computations involved. Designers of constrained systems must carefully balance storage efficiency and computational overhead based on system requirements and operational constraints. While constrained systems employ various key storage strategies, the decision to store full private keys or only seeds depends on design goals, performance considerations, and standards compliance (e.g., PKCS#11).

A challenge arises when importing an existing private key into a system designed to store only seeds. When a user attempts to import an already expanded private key, there is a mismatch between the key format used internally (seed-based) and the expanded private key. This issue arises because the internal format is designed for efficient key storage by deriving the private key from the seed, while the expanded private key is already fully computed. As NIST has not defined a single private key format for PQC algorithms, this creates a potential gap in interoperability.

If the seed is not securely stored at the time of key generation, it is permanently lost because the process of deriving an expanded key from the seed relies on a one-way cryptographic function. This one-way function derives the private key from the seed, but the reverse operation, deriving the original seed from the expanded key is computationally infeasible.

2.1.2. Efficient Key Derivation

When storing only the seed in a constrained cryptographic module, it is crucial that the device is capable of deriving the private key efficiently whenever required. However, it is important to note that constantly re-deriving the private key for every cryptographic operation may introduce significant performance overhead. In scenarios where performance is a critical consideration, it may be more efficient to store the expanded private key directly instead of only the seed. Higher quality implementations may also retain (cache) recently-used or frequently-used private keys to avoid the computational overhead and delay of deriving the private key from the seed with each request.

The key derivation process, such as ML-KEM.KeyGen_internal for ML-KEM or similar functions for other PQC algorithms, must be implemented in a way that can securely operate within the resource constraints of the device. If using the seed-only model, the derived private key should only be temporarily held in memory during the cryptographic operation and discarded immediately after use. However, storing the expanded private key may be a more practical solution in time-sensitive applications or for devices that frequently perform cryptographic operations.

2.1.3. Exporting Seeds and Private Keys

Given the potential for hardware failures or the end-of-life of devices containing keys, it is essential to plan for backup and recovery of the cryptographic seeds and private keys. Constrained devices should support secure seed backup mechanisms, ideally leveraging encrypted storage and ensuring that the backup data is protected from unauthorized access. In a disaster recovery scenario, the seeds and private keys should be recoverable private key, provided the proper security measures are in place to prevent unauthorized extraction.

There are two distinct approaches to exporting private keys or seeds from a constrained device:

2.1.3.1. Direct Transfer over TLS

In scenarios where the constrained device has sufficient capability to initiate or terminate a mutually authenticated TLS session, the device can securely transfer encrypted private key material directly to another cryptographic module.

2.1.3.2. Export to Encrypted File

In more common constrained device scenarios, for secure exporting of seeds and private keys, a strong symmetric encryption algorithm, such as AES, should be used to encrypt the seed before export. This ensures that the seed remains protected even if the export process is vulnerable to quantum attacks.

Operationally, the exported data and the AES key should both be protected against unauthorized access or modification.

2.1.3.3. Security Requirements for Export Operations

The encryption and decryption of seeds and private keys must occur entirely within the cryptographic modules to reduce the risk of exposure and ensure compliance to established security standards.

3. Ephemeral Key Management

In protocols like TLS and IPsec, ephemeral keys are used for key exchange. Given the increased size of PQC key material, ephemeral key management will have to be optimized for both security and performance.

For PQC KEMs, ephemeral key-pairs are generated from an ephemeral seed, that is used immediately during key generation and then discarded. Furthermore, once the shared secret is derived, the ephemeral private key will have to be deleted. Since the private key resides in the constrained cryptographic module, removing it optimizes memory usage, reducing the footprint of PQC key material in the cryptographic module. This ensures that that no unnecessary secrets persist beyond their intended use.

Additionally, ephemeral keys, whether from traditional ECDH or PQC KEM algorithms, are intended to be unique for each key exchange instance and kept separate across connections (e.g., TLS). Deleting ephemeral keying material after use not only optimizes memory usage but also ensures that key material cannot be reused across connections, which would otherwise introduce security and privacy issues. These risks are discussed in more detail in the Security Considerations of [I-D.ietf-tls-hybrid-design].

Constrained devices implementing PQC ephemeral key management will have to:

* Generate ephemeral key-pairs on-demand from an ephemeral seed stored temporarily within the cryptographic module.

* Enforce immediate seed erasure after the key-pair is generated and the cryptographic operation is completed.

* Delete the private key after the shared secret is derived.

* Prevent key reuse across different algorithm suites or sessions.

4. Optimizing Performance in Constrained Devices for PQC Signature Algorithms

When implementing PQC signature algorithms in constrained cryptographic modules, performance optimization becomes a critical consideration. Transmitting the entire message to the cryptographic module for signing can lead to significant overhead, especially for large payloads. To address this, implementers can leverage techniques that reduce the data transmitted to the cryptographic module, thereby improving efficiency and scalability.

One effective approach involves sending only a message digest to the cryptographic module for signing. By signing the digest of the content rather than the entire content, the communication between the application and the cryptographic module is minimized, enabling better performance. This method is applicable for any PQC signature algorithm, whether it is ML-DSA, SLH-DSA, or any future signature scheme. For such algorithms, a mechanism is often provided to pre-hash or process the message in a way that avoids sending the entire raw message for signing. In particular, algorithms like SLH-DSA present challenges due to their construction, which requires multiple passes over the message digest during the signing process. The signer does not retain the entire message or its full digest in memory at once. Instead, different parts of the message digest are processed sequentially during the signing procedure. This differs from traditional algorithms like RSA or ECDSA, which allow for more efficient processing of the message, without requiring multiple passes or intermediate processing of the digest.

A key consideration when deploying ML-DSA in cryptographic module is the amount of memory available. ML-DSA, unlike traditional signature schemes such as RSA or ECDSA, requires significant memory during signing due to multiple Number Theoretic Transform (NTT) operations, matrix expansions, and rejection sampling loops. These steps involve storing large polynomial vectors and intermediate values, making ML-DSA more memory-intensive. If an cryptographic module has sufficient memory, this may not be an issue. However, in constrained environments with limited memory, implementing ML-DSA can be challenging. The signer must store and process multiple transformed values, leading to increased computational overhead if the cryptographic module lacks the necessary memory to manage these operations efficiently.

To address the memory consumption challenge, algorithms like ML-DSA offer a form of pre-hash using the mu (message representative) value described in Section 6.2 of [ML-DSA]. The mu value provides an abstraction for pre-hashing by allowing the hash or message representative to be computed outside the cryptographic module. This feature offers additional flexibility by enabling the use of different cryptographic modules for the pre-hashing step, reducing memory consumption within the cryptographic module. The pre-computed mu value is then supplied to the cryptographic module, eliminating the need to transmit the entire message for signing. [I-D.ietf-lamps-dilithium-certificates] discusses leveraging ExternalMu-ML-DSA, where the pre-hashing step (ExternalMu-ML-DSA.Prehash) is performed in a software cryptographic module, and only the pre-hashed message (mu) is sent to the hardware cryptographic module for signing (ExternalMu-ML-DSA.Sign). By implementing ExternalMu-ML-DSA.Prehash in software and ExternalMu-ML-DSA.Sign in an hardware cryptographic module, the cryptographic workload is efficiently distributed, making it practical for high-volume signing operations even in memory-constrained cryptographic modules.

The main advantage of this method is that, unlike HashML-DSA, the ExternalMu-ML-DSA approach is interoperable with the standard version of ML-DSA that does not use pre-hashing. This means a message can be signed using ML-DSA.Sign, and the verifier can independently compute mu and use ExternalMu-ML-DSA.Verify for verification -- or vice versa. In both cases, the verifier does not need to know whether the signer used internal or external pre-hashing, as the resulting signature and verification process remain the same.

5. Additional Considerations for PQC Use in Constrained Devices

Key Rotation and Renewal: In constrained devices, managing the lifecycle of cryptographic keys including periodic key rotation and renewal is critical for maintaining long-term security and supporting cryptographic agility. While constrained devices may rely on integrated secure elements or lightweight HSMs for secure key storage and operations, the responsibility for orchestrating key rotation typically resides in the application layer or external device management infrastructure.

Although the underlying cryptographic module may offer primitives to securely generate new key pairs, store fresh seeds, or delete obsolete keys, these capabilities must be integrated into the device' s broader key management framework. This process is especially important in the context of PQC, where evolving research may lead to changes in recommended algorithms, parameters, and key management practices.

The security of PQC schemes continues to evolve, with potential risks arising from advances in post-quantum algorithms, cryptanalytic or implementation vulnerabilities. As a result, constrained devices should be designed to support flexible and updatable key management policies. This includes the ability to:

* Rotate keys periodically to provide forward-secrecy,

* Update algorithm choices or key sizes based on emerging security guidance,

* Reconfigure cryptographic profile of the device via firmware updates.

5.1. Key Sizes of Post-Quantum Algorithms

The key sizes of post-quantum algorithms are generally larger than those of traditional cryptographic algorithms. This increase in key size is a significant consideration for constrained devices, which often have limited memory and storage capacity. For example, the key sizes for ML-DSA and ML-KEM are larger than those of RSA or ECDSA, which can lead to increased memory usage and slower performance in constrained environments.

The following table provides the key sizes of some instantiations of ML-DSA, ML-KEM, FN-DSA and SLH-DSA. For comparision we also include the key sizes for X25519 and ED25519, which are traditional schemes widely used in constrained environments.

Algorithm	Type	Size (bytes)
ML-DSA-65	Public Key	1952
	Private Key	4032
	Signature	3309
SLH-DSA-SHA2-192s	Public Key	48
	Private Key	96
	Signature	16224
FN-DSA-512	Public Key	897
	Private Key	1281
	Signature	666
ML-KEM-768	Public Key	1568
	Shared Secret	32
X25519	Public Key	32
	Shared Secret	32
Ed25519	Public Key	32
	Signature	64

Table 1

Full key sizes for ML-DSA, ML-KEM, FN-DSA and SLH-DSA are specified in [ML-DSA], [ML-KEM], [FN-DSA] and [SLH-DSA] respectively.

6. Post-quantum Firmware Upgrades for Constrained Devices

Constrained devices deployed in the field require periodic firmware upgrades to patch security vulnerabilities, introduce new cryptographic algorithms, and improve overall functionality. However, the firmware upgrade process itself can become a critical attack vector if not designed to be post-quantum. If an adversary compromises the update mechanism, they could introduce malicious firmware, undermining all other security properties of the

cryptographic modules. Therefore, ensuring a post-quantum firmware upgrade process is critical for the security of deployed constrained devices.

CRQCs pose an additional risk by breaking traditional digital signatures (e.g., RSA, ECDSA) used to authenticate firmware updates. If firmware verification relies on traditional signature algorithms, attackers could generate forged signatures in the future and distribute malicious updates.

6.1. Post-quantum Firmware Authentication

To ensure the integrity and authenticity of firmware updates, constrained devices will have to adopt PQC digital signature schemes for code signing. These algorithms must provide long-term security, operate efficiently in low-resource environments, and be compatible with secure update mechanisms, such as the firmware update architecture for IoT described in [RFC9019].

The Software Updates for Internet of Things (SUIT) working group is defining mandatory-to-implement cryptographic algorithms for IoT devices in [I-D.ietf-suit-mti], which recommends use of HSS-LMS [RFC8554] to secure software devices.

Stateful hash-based signature schemes, such as HSS-LMS or the similar XMSS [RFC8391], are good candidates for signing firmware updates. Those schemes offer efficient verification times, making them more practical choices for constrained environments where performance and memory usage are key concerns. Their security is based on the security of the underlying hash function, which is well-understood. A major downside of stateful hash-based signatures is the requirement to keep track of which One-Time Signature (OTS) keys have been reused, since reuse of a single OTS key allows for signature forgeries. However, in the case of firmware updates, the OTS keys will be signing versioned updates, which may make state management easier. [I-D.ietf-pquip-hbs-state] discusses various strategies for a correct state and backup management for stateful hash-based signatures.

Other post-quantum signature algorithms may also be viable for firmware signing:

* SLH-DSA, a stateless hash-based signature specified in [SLH-DSA], also has well-understood security based on the security of its underlying hash function, and additionally doesn’t have the complexities associated with state management that HSS and XMSS have. However, signature generation and verification are comparatively slow, and signature sizes are generally larger than

other post-quantum algorithms. SLH-DSA's suitability as a firmware signing algorithm will depend on the capabilities of the underlying hardware.

* ML-DSA is a lattice-based signature algorithm specified in [ML-DSA]. It is more performant than SLH-DSA, with significantly faster signing and verification times, as well as shorter signatures. This will make it possible to implement on a wider range of constrained devices. The mathematical problem underpinning ML-DSA, Module Learning With Errors (M-LWE), is believed to be a hard problem by the cryptographic community, and hence ML-DSA is believed to be secure. Cryptographers are more confident still in the security of hash-based signatures than M-LWE, so developers may wish to factor that in when choosing a firmware signing algorithm.

6.2. Hybrid signature approaches

To enable secure migration from traditional to post-quantum security, hybrid signature methods can be used for firmware authentication. Parallel signatures, where a traditional and a post-quantum signature are generated and attached separately, is simple to implement, requires minimal changes to existing signing, and aligns well with current secure boot and update architectures.

Other hybrid techniques, such as cross-linked signatures (where signatures cover each other's values), composite signatures (which combine multiple signatures into a single structured signature), or counter-signatures (where one signature signs over another) introduce more complexity and are not yet typical in resource-constrained firmware workflows.

7. Impact of PQC Authentication on Constrained Devices

In constrained environments, devices are typically assumed to function as clients that initiate outbound connections, authenticating to servers using certificates or raw public keys ([RFC7250]). However, some devices also serve in server roles, enforcing local authentication policies. These scenarios require support for both outbound and inbound authentication, and both roles face significant challenges when adopting post-quantum cryptography (PQC). Additionally, verifying digital signatures such as during secure boot or firmware updates is a critical operation for constrained devices, regardless of whether they act as clients or servers.

While specific deployment scenarios may differ, the fundamental technical impacts of PQC authentication in constrained devices can be summarized into three main areas:

* Larger Signatures and Certificate Sizes

  Post-quantum signature schemes typically produce much larger public keys and signatures than their traditional counterparts. A comparison is provided in Section 5.1.

  These larger artifacts introduce several challenges. For example, certificate chains with PQC public keys require more storage, and trust anchors - particularly for schemes like SLH-DSA - may be too large to embed in constrained ROM.

  Furthermore, validating signed payloads or commands increases network bandwidth requirements. In the case of large hash-based signatures, implementations may adopt streaming verification, where only parts of the message are processed at a time to reduce memory usage. An example of such an approach for SLH-DSA is described in [Stream-SPHINCS].

* Increased RAM usage and performance profile.

  Post-quantum signature verification often demands significantly more RAM than traditional schemes used for asymmetric cryptography. For example, ML-DSA-65 in its high-performance configuration may require over 68KB of memory during signing and up to 10 KB during verification on Cortex-M4-class devices.

  This poses challenges for use cases such as firmware verification (e.g. secure boot) and certificate validation during TLS handshakes or the generation of signed claims about the devices's hardware and software state, a process generally referred to as device attestation. As part of this remote attestation procedure [RFC9334], the device will need to present such claims to a remote peer, signed using an attestation key. To remain secure against CRQCs, the attestation mechanism must also employ quantum-safe cryptographic primitives.

  Several memory-optimized implementations exist (see [BosRS22]), but they typically trade memory savings for slower performance. For instance, the ML-DSA.Sign operation can be implemented within 8KB of RAM, though at the cost of significantly increased runtime. Conversely, ML-DSA.Verify can be performed in as little as 3KB of RAM without a major performance penalty.

Devices with 8 - 16KB of available RAM must often balance performance against feasibility when integrating PQC signature verification.

When constrained devices must authenticate inbound connections, validate commands, or verify stored data, PQC authentication imposes a burden that must be explicitly addressed through selection of schemes with smaller signature sizes (e.g. FN-DSA). These choices should be aligned with the device's operational profile, available memory, and longevity requirements.

8. Security Considerations

The security considerations for key management in constrained devices for PQC focus on the secure storage and handling of cryptographic seeds, which are used to derive private keys. Seeds must be protected with the same security measures as private keys, and key derivation should be efficient and secure within resource-constrained cryptographic module. Secure export and backup mechanisms for seeds are essential to ensure recovery in case of hardware failure, but these processes must be encrypted and protected from unauthorized access.

8.1. Side Channel Protection

Side-channel attacks exploit physical leaks during cryptographic operations, such as timing information, power consumption, electromagnetic emissions, or other physical characteristics, to extract sensitive data like private keys or seeds. Given the sensitivity of the seed and private key in PQC key generation, it is critical to consider side-channel protection in cryptographic module design. While side-channel attacks remain an active research topic, their significance in secure hardware design cannot be understated. Cryptographic modules must incorporate strong countermeasures against side-channel vulnerabilities to prevent attackers from gaining insights into secret data during cryptographic operations.

Acknowledgements

Thanks to Jean-Pierre Fiset, Richard Kettlewell, Mike Ounsworth, and Aritra Banerjee for the detailed review.

References

Normative References

[RFC7250] Wouters, P., Ed., Tschofenig, H., Ed., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7250, DOI 10.17487/RFC7250, June 2014, <https://www.rfc-editor.org/rfc/rfc7250>.

[RFC9019] Moran, B., Tschofenig, H., Brown, D., and M. Meriac, "A Firmware Update Architecture for Internet of Things", RFC 9019, DOI 10.17487/RFC9019, April 2021, <https://www.rfc-editor.org/rfc/rfc9019>.

[RFC9334] Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote ATtestation procedureS (RATS) Architecture", RFC 9334, DOI 10.17487/RFC9334, January 2023, <https://www.rfc-editor.org/rfc/rfc9334>.

Informative References

[BIND] "Unbindable Kemmy Schmid", n.d., <https://eprint.iacr.org/2024/523.pdf>.

[BosRS22] "Dilithium for Memory Constrained Devices", <https://eprint.iacr.org/2022/323.pdf>.

[FN-DSA] "Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU", <https://falcon-sign.info/falcon.pdf>.

[I-D.ietf-lamps-dilithium-certificates] Massimo, J., Kampanakis, P., Turner, S., and B. Westerbaan, "Internet X.509 Public Key Infrastructure - Algorithm Identifiers for the Module-Lattice-Based Digital Signature Algorithm (ML-DSA)", Work in Progress, Internet-Draft, draft-ietf-lamps-dilithium-certificates-12, 26 June 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-lamps-dilithium-certificates-12>.

[I-D.ietf-lamps-kyber-certificates] Turner, S., Kampanakis, P., Massimo, J., and B. Westerbaan, "Internet X.509 Public Key Infrastructure - Algorithm Identifiers for the Module-Lattice-Based Key-Encapsulation Mechanism (ML-KEM)", Work in Progress, Internet-Draft, draft-ietf-lamps-kyber-certificates-10, 16 April 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-lamps-kyber-certificates-10>.

[I-D.ietf-pquip-hbs-state] Wiggers, T., Bashiri, K., Klbl, S., Goodman, J., and S. Kousidis, "Hash-based Signatures: State and Backup

Management", Work in Progress, Internet-Draft, draft-ietf-pquip-hbs-state-00, 17 June 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-pquip-hbs-state-00>.

[I-D.ietf-suit-mti]
Moran, B., Rnningstad, O., and A. Tsukamoto, "Cryptographic Algorithms for Internet of Things (IoT) Devices", Work in Progress, Internet-Draft, draft-ietf-suit-mti-20, 4 July 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-suit-mti-20>.

[I-D.ietf-tls-hybrid-design]
Stebila, D., Fluhrer, S., and S. Gueron, "Hybrid key exchange in TLS 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-hybrid-design-13, 17 June 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-hybrid-design-13>.

[ML-DSA] "FIPS-204: Module-Lattice-Based Digital Signature Standard", <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.204.pdf>.

[ML-KEM] "FIPS-203: Module-Lattice-based Key-Encapsulation Mechanism Standard", <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.203.pdf>.

[REC-KEM] "Recommendations for Key-Encapsulation Mechanisms", n.d., <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-227.ipd.pdf>.

[REC-SHS] "Recommendation for Stateful Hash-Based Signature Scheme", <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-208.pdf>.

[RFC8391] Huelsing, A., Butin, D., Gazdag, S., Rijneveld, J., and A. Mohaisen, "XMSS: eXtended Merkle Signature Scheme", RFC 8391, DOI 10.17487/RFC8391, May 2018, <https://www.rfc-editor.org/rfc/rfc8391>.

[RFC8554] McGrew, D., Curcio, M., and S. Fluhrer, "Leighton-Micali Hash-Based Signatures", RFC 8554, DOI 10.17487/RFC8554, April 2019, <https://www.rfc-editor.org/rfc/rfc8554>.

[SLH-DSA] "FIPS-205: Stateless Hash-Based Digital Signature Standard", <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.205.pdf>.

[Stream-SPHINCS] "Streaming SPHINCS+ for Embedded Devices using the Example of TPMs", <https://eprint.iacr.org/2021/1072.pdf>.

Authors' Addresses

Tirumaleswar Reddy
Nokia
Bangalore
Karnataka
India
Email: k.tirumaleswar_reddy@nokia.com

Dan Wing
Cloud Software Group Holdings, Inc.
United States of America
Email: danwing@gmail.com

Ben Salter
UK National Cyber Security Centre
Email: ben.s3@ncsc.gov.uk

Kris Kwiatkowski
PQShield
Email: kris@amongbytes.com