SSHM
Internet-Draft
Intended status: Informational
Expires: 20 June 2026

P. Kampanakis
AWS
D. Stebila
University of Waterloo
T. Hansen
AWS

17 December 2025

PQ/T Hybrid Key Exchange with ML-KEM in SSH
draft-ietf-sshm-mlkem-hybrid-kex-07

Abstract

This document defines Post-Quantum Traditional (PQ/T) Hybrid key exchange methods based on the quantum-resistant the Module-Lattice-Based Key-Encapsulation Mechanism (ML-KEM) standard and traditional Elliptic-curve Diffie宴滴ellman (ECDH) key exchange schemes. These methods are defined for use in the SSH Transport Layer Protocol.

Table of Contents

1. Introduction

Secure Shell (SSH) [RFC4251] performs key establishment using key exchange methods based on (Elliptic Curve) Diffie-Hellman style schemes defined in [RFC5656] and [RFC8731]. The cryptographic security of these key exchanges relies on certain instances of the discrete logarithm problem being computationally infeasible to solve for adversaries.

However, if sufficiently large quantum computers become available, these instances would no longer be computationally infeasible rendering the current key exchange and authentication methods in SSH insecure [I-D.hoffman-c2pq]. While large quantum computers are not available today an adversary could record the encrypted communication sent between the client and server in an SSH session and later

decrypt it when sufficiently large quantum computers become available. This kind of attack is known as a "harvest-now-decrypt-later" attack.

This document addresses the problem by extending the SSH Transport Layer Protocol [RFC4253] key exchange with Post-Quantum Traditional (PQ/T) Hybrid [RFC9794] key exchange methods. The security provided by each key exchange scheme in a PQ/T Hybrid key exchange method is independent. This means that the PQ/T Hybrid key exchange method will always be at least as secure as the most secure key exchange scheme executed as part of the exchange. [PQ-PROOF] [PQ-PROOF2] contain proofs of security for such PQ/T Hybrid key exchange schemes.

In the context of the [NIST_PQ], key exchange algorithms are formulated as key encapsulation mechanisms (KEMs), which consist of three algorithms:

* 'KeyGen() -> (pk, sk)': A probabilistic key generation algorithm, which generates a public key 'pk' and a secret key 'sk'.

* 'Encaps(pk) -> (ct, ss)': A probabilistic encapsulation algorithm, which takes as input a public key 'pk' and outputs a ciphertext 'ct' and shared secret 'ss'.

* 'Decaps(sk, ct) -> ss': A decapsulation algorithm, which takes as input a secret key 'sk' and ciphertext 'ct' and outputs a shared secret 'ss', or in some cases a distinguished error value.

The main security property for KEMs is indistinguishability under adaptive chosen ciphertext attack (IND-CCA2), which means that shared secret values should be indistinguishable from random strings even given the ability to have arbitrary ciphertexts decapsulated. IND-CCA2 corresponds to security against an active attacker, and the public key / secret key pair can be treated as a long-term key or reused. A weaker security notion is indistinguishability under chosen plaintext attack (IND-CPA), which means that the shared secret values should be indistinguishable from random strings given a copy of the public key. IND-CPA roughly corresponds to security against a passive attacker, and sometimes corresponds to one-time key exchange.

The post-quantum KEM used in the document is ML-KEM. ML-KEM was standardized in 2024 [FIPS203] with three parameter variants, ML-KEM-512, ML-KEM-768, and ML-KEM-1024. This specification's PQ/T Hybrid key exchange message abstraction, key derivation, and input to the SSH hash calculation, H, align with the ones defined in [I-D.ietf-sshm-ntruprime-ssh] which uses a different quantum-resistant KEM.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC8174] [RFC2119] when, and only when, they appear in all capitals, as shown here.

2. PQ/T Hybrid Key Exchange

2.1. PQ/T Hybrid Key Exchange Method Abstraction

This section defines the abstract structure of a PQ/T Hybrid key exchange method. This structure must be instantiated with two key exchange schemes. The byte and string types are to be interpreted in this document as described in [RFC4251].

In a PQ/T Hybrid key exchange, instead of SSH_MSG_KEXDH_INIT [RFC4253] or SSH_MSG_KEX_ECDH_INIT [RFC5656], the client sends

```
byte     SSH_MSG_KEX_HYBRID_INIT
string   C_INIT
```

where C_INIT is the concatenation of C_PK2 and C_PK1 (C_INIT = C_PK2 || C_PK1, where || depicts concatenation). C_PK1 and C_PK2 represent the ephemeral client public keys used for each key exchange of the PQ/T Hybrid mechanism. Typically, C_PK1 represents a traditional / classical (i.e., ECDH) key exchange public key. C_PK2 represents the ’pk’ output of the corresponding post-quantum KEM’s ’KeyGen’ at the client.

Instead of SSH_MSG_KEXDH_REPLY [RFC4253] or SSH_MSG_KEX_ECDH_REPLY [RFC5656], the server sends

```
byte     SSH_MSG_KEX_HYBRID_REPLY
string   K_S, server’s public host key
string   S_REPLY
string   the signature on the exchange hash
```

where S_REPLY is the concatenation of S_CT2 and S_PK1 (S_REPLY = S_CT2 || S_PK1). Typically, S_PK1 represents the ephemeral (EC)DH server public key. S_CT2 represents the ciphertext ’ct’ output of the corresponding KEM’s ’Encaps’ algorithm generated by the server which encapsulates a secret to the client’s public key C_PK2. Before producing S_CT2, to prevent length extension attack attempts, the server MUST check that the length of the C_INIT is the sum of the expected length of each public key in the negotiated method, C_PK1 and C_PK2. It also MUST perform the encapsulation key checks defined

in Section 7.2 of [FIPS203]. If any of these checks fail, the client MUST abort using a disconnect message (SSH_MSG_DISCONNECT) with a SSH_DISCONNECT_KEY_EXCHANGE_FAILED as the reason.

C_PK1, S_PK1, C_PK2, and S_CT2 are used to establish two shared secrets, K_CL and K_PQ. K_CL is the output from the classical ECDH exchange using C_PK1 and S_PK1. K_PQ is the post-quantum shared secret decapsulated from S_CT2. Before decapsulating, to prevent length extension attack attempts, the client MUST check that the length of the S_REPLY is the sum of the expected length of the traditional public key, S_PK1, and the ML-KEM ciphertext, S_CT2, in the negotiated method. The client MUST abort using a disconnect message (SSH_MSG_DISCONNECT) with a SSH_DISCONNECT_KEY_EXCHANGE_FAILED as the reason if the check fails or decapsulation fails for any other reason. K_CL and K_PQ are used together to generate the shared secret K according to Section 2.4.

For all method names, both the client and server MUST process the ECDH and X25519 public keys (C_PK1, S_PK1) as described in Section 4 of [RFC5656] and Section 3 of [RFC8731] respectively, including validity and length checks and SSH disconnect messages if the checks fail.

2.2. PQ/T Hybrid Key Exchange Message Numbers

The message numbers 30-49 are key-exchange-specific and in a private namespace defined in [RFC4250] that may be redefined by any key exchange method [RFC4253] without requiring an IANA registration process.

The following private namespace message numbers are defined in this document:

```
#define SSH_MSG_KEX_HYBRID_INIT              30
#define SSH_MSG_KEX_HYBRID_REPLY             31
```

2.3. PQ/T Hybrid Key Exchange Method Names

The PQ/T Hybrid key exchange method names defined in this document (to be used in SSH_MSG_KEXINIT [RFC4253]) are

```
mlkem768nistp256-sha256
mlkem1024nistp384-sha384
mlkem768x25519-sha256
```

These instantiate the abstract PQ/T Hybrid key exchanges defined in Section 2.1.

2.3.1. mlkem768nistp256-sha256

mlkem768nistp256-sha256 defines that the traditional client and server public keys C_PK1, S_PK1 belong to the NIST P-256 curve [nist-sp800-186]. The private and public keys are generated as described therein. The public keys are defined as octet strings for NIST P-256 as per [RFC5656]; point compression may be used. The K_CL shared secret is generated from the exchanged C_PK1 and S_PK1 public keys as defined in [RFC5656] (key agreement method ecdh-sha2-nistp256).

The post-quantum C_PK2 and S_CT2 represent ML-KEM-768 public key and ciphertext from the client and server respectively which are encoded as octet strings. The K_PQ shared secret is decapsulated from the ciphertext S_CT2 using the client post-quantum KEM private key as defined in [FIPS203].

The HASH function used in the key exchange [RFC4253] is SHA-256 [nist-sha2] [RFC6234].

2.3.2. mlkem1024nistp384-sha384

mlkem1024nistp384-sha384 defines that the traditional client and server public keys C_PK1, S_PK1 belong to the NIST P-384 curve [nist-sp800-186]. The private and public keys are generated as described therein. The public keys are defined as octet strings for NIST P-384 as per [RFC5656]; point compression may be used. The K_CL shared secret is generated from the exchanged C_PK1 and S_PK1 public keys as defined in [RFC5656] (key agreement method ecdh-sha2-nistp384).

The post-quantum C_PK2 and S_CT2 represent ML-KEM-1024 public key and ciphertext from the client and server respectively which are encoded as octet strings. The K_PQ shared secret is decapsulated from the ciphertext S_CT2 using the client post-quantum KEM private key as defined in [FIPS203].

The HASH function used in the key exchange [RFC4253] is SHA-384 [nist-sha2] [RFC6234].

2.3.3. mlkem768x25519-sha256

mlkem768x25519-sha256 defines that the traditional client and server public keys C_PK1, S_PK1 belong to the Curve25519 curve [RFC7748]. Private and public keys are generated as described therein. The public keys are defined as strings of 32 bytes as per [RFC8731]. The K_CL shared secret is generated from the exchanged C_PK1 and S_PK1 public keys as defined in [RFC8731] (key agreement method

curve25519-sha256).

The post-quantum C_PK2 and S_CT2 represent ML-KEM-768 public key and ciphertext from the client and server respectively which are encoded as octet strings. The K_PQ shared secret is decapsulated from the ciphertext S_CT2 using the client post-quantum KEM private key as defined in [FIPS203].

The HASH function used in the key exchange [RFC4253] is SHA-256 [nist-sha2] [RFC6234].

2.4. Shared Secret K

The PQ/T Hybrid key exchange establishes K_CL and K_PQ from the ECDH and ML-KEM key exchanges respectively. The shared secret, K, is the HASH output of the concatenation of the two shared secrets K_CL and K_PQ as

```
K = HASH(K_PQ || K_CL)
```

This is similar, but not the same (for efficiency), logic as in TLS 1.3 [I-D.ietf-tls-hybrid-design]. In [I-D.ietf-tls-hybrid-design], the classical and post-quantum exchanged secrets are concatenated and used in the key schedule whereas in this document they are concatenated and hashed before being used in SSH’s key derivation methodology.

The ECDH shared secret was traditionally encoded as an integer (mpint) as per [RFC4253], [RFC5656], and [RFC8731] and used in deriving the key. In this specification, the two shared secrets, K_PQ and K_CL, are fed into the hash function to derive K, but they are encoded as fixed-length byte arrays, not as integers. Byte arrays are defined in Section 5 of [RFC4251]. Specifically for K_CL, the conversion from mpint to a byte array is done by taking the mpint that the corresponding standalone key exchange method would have output and re-encoding it as a fixed-size (32 bytes for Curve25519 and secp256r1 or 48 bytes for secp384r1) byte array always big-endian.

2.5. Key Derivation

The derivation of encryption keys MUST be done from the shared secret K according to Section 7.2 in [RFC4253] with a modification on the exchange hash H.

The PQ/T Hybrid key exchange hash H is the result of computing the HASH, where HASH is the hash algorithm specified in the named PQ/T Hybrid key exchange method name, over the concatenation of the following

```
string V_C, client identification string (CR and LF excluded)
string V_S, server identification string (CR and LF excluded)
string I_C, payload of the client's SSH_MSG_KEXINIT
string I_S, payload of the server's SSH_MSG_KEXINIT
string K_S, server's public host key
string C_INIT, client message octet string
string S_REPLY, server message octet string
string K, SSH shared secret
```

K, the shared secret used in H, was traditionally encoded as an integer (mpint) as per [RFC4253], [RFC5656], and [RFC8731]. In this specification, K is the hash output of the two concatenated byte arrays (Section 2.4) which is not an integer. Thus, K is encoded as a string using the process described in Section 5 of [RFC4251] and is then fed along with other data in H to the key exchange method's HASH function to generate encryption keys.

3. Message Size

An implementation adhering to [RFC4253] must be able to support packets with an uncompressed payload length of 32768 bytes or less and a total packet size of 35000 bytes or less (including 'packet_length', 'padding_length', 'payload', 'random padding', and 'mac'). These numbers represent what must be 'minimally supported' by implementations. This can present a problem when using post-quantum key exchange schemes because some post-quantum schemes can produce much larger messages than what is normally produced by existing key exchange methods defined for SSH. This document does not define any method names (Section 2.3) that cause any PQ/T Hybrid key exchange method related packets to exceed the minimally supported packet length. This document does not define behavior in cases where a PQ/T Hybrid key exchange message causes a packet to exceed the minimally supported packet length.

4. Acknowledgements

The authors want to thank Gerardo Ravago from AWS for implementing the draft and finding issues. We also want to thank Damien Miller and Markus Friedl for their feedback and for implementing some of the SSH key exchange methods in this document in OpenSSH. Special acknowledgements go to Simon Tatham from Putty, Loganaden Velvindron, John Mattsson, Simon Josefsson, and Watson Ladd for their valuable suggestions.

5. IANA Considerations

This memo requests IANA to register new method names "mlkem768nistp256-sha256", "mlkem1024nistp384-sha384", and "mlkem768x25519-sha256" in the "Key Exchange Method Names" registry for SSH [IANA-SSH] with a "Reference" field to this RFC and the "OK to implement" field of "SHOULD".

6. Security Considerations

The security considerations given in [RFC5656] and [RFC8731] also apply to the ECDH part of the P/T Hybrid key exchange schemes defined in this document.

As it is commonly done with (EC)DH keys today, generating an ephemeral key exchange keypair for ECDH and ML-KEM per connection is REQUIRED by this specification. Additionally, implementations MUST NOT reuse randomness in the generation of ML-KEM ciphertexts. As a reminder, the security properties of the protocol in this document, SSH itself, and the cryptographic algorithms used, including ML-KEM, depend on the availability and proper use of cryptographically secure random data. [RFC4086] includes guidance regarding randomness for security.

Implementations MUST use the encodings for K_PQ, K_CL, and K specified in this document to prevent potential side-channel attacks. The way a derived binary secret string is encoded (i.e., adding or removing zero bytes for encoding) before it is hashed may lead to a variable-length secret which raises the potential for a side-channel attack. In broad terms, when the secret is longer, the hash function may need to process more blocks internally which could determine the length of what is hashed. This could leak the most significant bit of the derived secret and/or allow detection of when the most significant bytes are zero. In some unfortunate circumstances, this has led to timing attacks, e.g. the Lucky Thirteen [LUCKY13] and Raccoon [RACCOON] attacks. In [RFC8731] and [RFC5656], the ECDH shared secrets were mpint and fixed-length integer encoded respectively which raised a potential for such side-channel attacks. This problem is addressed in this document by encoding K_PQ and K_CL as fixed-length byte arrays and K as a string.

[PQ-PROOF] [PQ-PROOF2] contain proofs of security for PQ/T Hybrid key exchange schemes. [PQ-PROOF2] discusses how the key combination to derive K and the derivation of SSH symmetric keys in this document can be proven IND-CPA and IND-CCA2 secure with some assumptions. IND-CPA is achieved if we assume the HASH calls perform as a KDF which is a reasonable assumption. IND-CCA2 security is achieved by assuming the HASH is a random oracle which is a stronger assumption

especially for variants of the SHA-2 family which introduce length extension risks. To leverage a HASH which is more suitable as a random oracle, we could use SHAKE256 or introduce HMAC-SHA-256 as proposed in options (2b) and (2c) in Appendix A. This document uses SHA-2 which is ubiquitous although it makes an IND-CCA2 proof need stronger assumptions because even SSH's traditional key derivation has not been proven to be IND-CCA2.

X25519, the traditional elliptic curve key exchange used in one of the PQ/T hybrid methods specified in this document, is generally considered easier to implement securely without side-channels than its NIST counterparts (with P256, P384). Historically, implementations of P256 and P384 have suffered various implementation issues which have been addressed over time. Optimized X25119 implementations are also more efficient than P256 and P384. Thus, X25519 has seen more adoption than P256 and P384 across cryptographic use-cases. NIST curves are sometimes preferred for regulatory compliance.

7. References

7.1. Normative References

[FIPS203] National Institute of Standards and Technology (NIST), "Module-Lattice-Based Key-Encapsulation Mechanism Standard", NIST Federal Information Processing Standards, 13 August 2024, <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.203.pdf>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC4251] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Protocol Architecture", RFC 4251, DOI 10.17487/RFC4251, January 2006, <https://www.rfc-editor.org/info/rfc4251>.

[RFC4253] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, DOI 10.17487/RFC4253, January 2006, <https://www.rfc-editor.org/info/rfc4253>.

[RFC5656] Stebila, D. and J. Green, "Elliptic Curve Algorithm Integration in the Secure Shell Transport Layer", RFC 5656, DOI 10.17487/RFC5656, December 2009, <https://www.rfc-editor.org/info/rfc5656>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8731] Adamantiadis, A., Josefsson, S., and M. Baushke, "Secure Shell (SSH) Key Exchange Method Using Curve25519 and Curve448", RFC 8731, DOI 10.17487/RFC8731, February 2020, <https://www.rfc-editor.org/info/rfc8731>.

7.2. Informative References

[I-D.connolly-cfrg-xwing-kem]
Connolly, D., Schwabe, P., and B. Westerbaan, "X-Wing: general-purpose hybrid post-quantum KEM", Work in Progress, Internet-Draft, draft-connolly-cfrg-xwing-kem-09, 1 September 2025, <https://datatracker.ietf.org/doc/html/draft-connolly-cfrg-xwing-kem-09>.

[I-D.hoffman-c2pq]
Hoffman, P. E., "The Transition from Classical to Post-Quantum Cryptography", Work in Progress, Internet-Draft, draft-hoffman-c2pq-07, 26 May 2020, <https://datatracker.ietf.org/doc/html/draft-hoffman-c2pq-07>.

[I-D.ietf-sshm-ntruprime-ssh]
Friedl, M., Mojzis, J., and S. Josefsson, "Secure Shell (SSH) Key Exchange Method Using Hybrid Streamlined NTRU Prime sntrup761 and X25519 with SHA-512: sntrup761x25519-sha512", Work in Progress, Internet-Draft, draft-ietf-sshm-ntruprime-ssh-06, 30 September 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-sshm-ntruprime-ssh-06>.

[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-16, 7 September 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-hybrid-design-16>.

[I-D.josefsson-chempat]
Josefsson, S., "Chempat: Generic Instantiated PQ/T Hybrid Key Encapsulation Mechanisms", Work in Progress, Internet-Draft, draft-josefsson-chempat-04, 20 October 2025, <https://datatracker.ietf.org/doc/html/draft-josefsson-chempat-04>.

[IANA-SSH] IANA, "Secure Shell (SSH) Protocol Parameters", 2021, <https://www.iana.org/assignments/ssh-parameters/ssh-parameters.xhtml>.

[LUCKY13] Al Fardan, N.J. and K.G. Paterson, "Lucky Thirteen: Breaking the TLS and DTLS record protocols", 2013, <https://ieeexplore.ieee.org/iel7/6547086/6547088/06547131.pdf>.

[nist-sha2] NIST, "FIPS PUB 180-4", 2015, <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf>.

[NIST-SP-800-133r2] National Institute of Standards and Technology (NIST), "Recommendation for Cryptographic Key Generation", June 2020, <https://doi.org/10.6028/NIST.SP.800-133r2>.

[NIST-SP-800-135] National Institute of Standards and Technology (NIST), "Recommendation for Existing Application-Specific Key Derivation Functions", December 2011, <https://doi.org/10.6028/NIST.SP.800-135r1>.

[NIST-SP-800-227] National Institute of Standards and Technology (NIST), "Recommendations for Key-Encapsulation Mechanisms", September 2025, <https://doi.org/10.6028/NIST.SP.800-227>.

[NIST-SP-800-56C] National Institute of Standards and Technology (NIST), "Recommendation for Key-Derivation Methods in Key-Establishment Schemes", August 2020, <https://doi.org/10.6028/NIST.SP.800-56Cr2>.

[nist-sp800-186] NIST, "SP 800-186", 2019, <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-186-draft.pdf>.

[NIST_PQ] NIST, "Post-Quantum Cryptography", 2020, <https://csrc.nist.gov/projects/post-quantum-cryptography>.

[PQ-PROOF] Campagna, M. and A. Petcher, "Security of Hybrid Key Encapsulation", 2020, <https://eprint.iacr.org/2020/1364>.

[PQ-PROOF2] Petcher, A. and M. Campagna, "Security of Hybrid Key Establishment using Concatenation", 2023, <https://eprint.iacr.org/2023/972>.

[RACCOON] Merget, R., Brinkmann, M., Aviram, N., Somorovsky, J., Mittmann, J., and J. Schwenk, "Raccoon Attack: Finding and Exploiting Most-Significant-Bit-Oracles in TLS-DH(E)", September 2020, <https://raccoon-attack.com/>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <https://www.rfc-editor.org/info/rfc4086>.

[RFC4250] Lehtinen, S. and C. Lonvick, Ed., "The Secure Shell (SSH) Protocol Assigned Numbers", RFC 4250, DOI 10.17487/RFC4250, January 2006, <https://www.rfc-editor.org/info/rfc4250>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <https://www.rfc-editor.org/info/rfc6234>.

[RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <https://www.rfc-editor.org/info/rfc7748>.

[RFC9794] Driscoll, F., Parsons, M., and B. Hale, "Terminology for Post-Quantum Traditional Hybrid Schemes", RFC 9794, DOI 10.17487/RFC9794, June 2025, <https://www.rfc-editor.org/info/rfc9794>.

Appendix A. Other Combiners

Other combiners to derive K and the SSH keys were considered while working on this document. These include

(1) K = K_PQ || K_CL. All SSH keys are derived from K as defined in Section 7.2 in [RFC4253].

(2) All SSH keys are derived from K as defined in Section 7.2 in [RFC4253].

    (a) K = HASH(K_PQ, K_CL). This is the option adopted in this specification.

(b) K = HMAC-HASH(K_PQ, K_CL)

(c) K = HMAC-HASH(0, K_PQ || K_CL)

(3) K = HKDF-HASH_Extract(0, K_PQ || K_CL). SSH keys are now derived from K using HKDF-HASH(K, H || session_id, 6*sizeof(HASH)).

Option (3) follows the Extract-and-Expand logic described in [NIST-SP-800-56C]. It deviates from existing SSH key derivation significantly and might be viewed as too far from the current SSH design. It probably would be a good approach for SSH to move from basic hashing everywhere to use proper KDFs with extract/expand, but that should be a separate effort.

We also considered combiners like the ones proposed in [I-D.josefsson-chempat] and [I-D.connolly-cfrg-xwing-kem]. [I-D.connolly-cfrg-xwing-kem] has a separate IND-CCA2 security proof. Although such combiners may be proven IND-CCA2 secure, to be IND-CCA2, the SSH key derivation would still require the assumptions laid out in [PQ-PROOF2] and discussed in Section 6.

Appendix B. FIPS

[NIST-SP-800-56C] and [NIST-SP-800-135] give NIST recommendations for key derivation methods in key exchange protocols. Some PQ/T Hybrid combinations may combine the shared secret from a NIST-approved algorithm (e.g., ECDH using the nistp256/secp256r1 curve or ML-KEM) with a shared secret from a non-approved algorithm (e.g., X25519). [NIST-SP-800-227] lists simple concatenation as an approved way of producing a PQ/T Hybrid shared secret in which one of the constituent secrets is from an approved algorithm (i.e., secp256r1, secp384r1, ML-KEM) and using it in a key derivation/combination method approved by [NIST-SP-800-56C] or [NIST-SP-800-133r2]. Although the SSH key derivation function does not follow [NIST-SP-800-56C] or [NIST-SP-800-133r2], it is approved by [NIST-SP-800-135]. This method is the same used in this document to derive SSH keys from the quantum-resistant shared secret. Thus, the SSH key combiner in this document appears to be FIPS-approved although it is not specifically called out in [NIST-SP-800-227].

Authors' Addresses

Panos Kampanakis
AWS
Email: kpanos@amazon.com

Douglas Stebila
University of Waterloo
Email: dstebila@uwaterloo.ca

Torben Hansen
AWS
Email: htorben@amazon.com