Workgroup: Network Working Group
Internet-Draft:
draft-kampanakis-curdle-ssh-pq-ke-01
Published: 10 April 2023
Intended Status: Experimental
Expires: 12 October 2023
Authors: P. Kampanakis
AWS

D. Stebila
University of Waterloo

T. Hansen
AWS

# Post-quantum Hybrid Key Exchange in SSH

## Abstract

This document defines post-quantum hybrid key exchange methods based on classical ECDH key exchange and post-quantum key encapsulation schemes. These methods are defined for use in the SSH Transport Layer Protocol.

[EDNOTE: Discussion of this work is encouraged to happen on the IETF WG Mailing List or in the GitHub repository which contains the draft: https://github.com/csosto-pk/pq-ssh/issues.]

## Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 12 October 2023.

**Table of Contents**



## 1. Introduction

Change Log [EDNOTE: Remove before publicaton. ]
* draft-kampanakis-curdle-ssh-pq-ke-00
  Initial draft replacing draft-kampanakis-curdle-pq-ssh-00

Secure Shell (SSH) RFC4251 [RFC4251] performs key establishment using key exchange methods based on (Elliptic Curve) Diffie-Hellman style schemes. SSH [RFC4252] [RFC8332] [RFC5656] [RFC8709] also defines public key authentication methods based on RSA, ECDSA, or EdDSA signature schemes. The cryptographic security of these key exchange and signature schemes relies on certain instances of the discrete logarithm and integer factorization problems 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 "record-and-harvest" attack.

This document addresses the problem by extending the SSH Transport Layer Protocol RFC4253 [RFC4253] key exchange with post-quantum (PQ) hybrid (PQ-hybrid) key exchange methods. The security provided by each individual key exchange scheme in a PQ-hybrid key exchange method is independent. This means that the PQ-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] contains proofs of security for such PQ-hybrid key exchange schemes.

In the context of the NIST Post-Quantum Cryptography Standardization Project [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.

### 1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

## 2. PQ-hybrid Key Exchange

### 2.1. PQ-hybrid Key Exchange Method Abstraction

This section defines the abstract structure of a PQ-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 [RFC4251].

In a PQ-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_PK1 and C_PK2 represent the ephemeral client public keys used for each key exchange of the PQ-hybrid mechanism. Typically, C_PK1 represents a 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. 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 public key C_PK2.

[EDNOTE: Initially we were encoding the server and client client and server classical and post-quantum public key/ciphertext as its own string. We since switched to an encoding method which concatenates them together as a single string in the C_INIT, S_REPLY message. This method concatenates the raw values rather than the length of each value plus the value. The total length of the concatenation is still known, but the relative lengths of the individual values that were concatenated is no longer part of the representation. This assumes that the lengths of individual values are fixed once the algorithm is selected, which is the case for classical key exchange methods currently supported by SSH and all post-quantum KEMs in Round 3 of the NIST post-quantum standardization project. If that is the WG consensus we need to put a note of this in the Appendix for

historical reference and expand on the concatenated string here in this section.]

C_PK1, S_PK1, C_PK2, 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. K_CL and K_PQ are used together to generate the shared secret K according to Section 2.4.

### 2.2. PQ-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 message numbers have been defined in this document:

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

### 2.3. PQ-hybrid Key Exchange Method Names

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

```
ecdh-nistp256-kyber-512r3-sha256-d00@openquantumsafe.org
ecdh-nistp384-kyber-768r3-sha384-d00@openquantumsafe.org
ecdh-nistp521-kyber-1024r3-sha512-d00@openquantumsafe.org
x25519-kyber-512r3-sha256-d00@amazon.com
```

These instantiate abstract PQ-hybrid key exchanges defined in Section 2.1.

#### 2.3.1. ecdh-nistp256-kyber-512r3-sha256-d00@openquantumsafe.org

ecdh-nistp256-kyber-512r3-sha256-d00@openquantumsafe.org defines that the classical 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 Kyber512 public key and ciphertext from the 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 [EDNOTE: Add PQ KEM specification link here].

The HASH function used in the key exchange [RFC4253] is SHA-256 [nist-sha2] [RFC4634][EDNOTE: Keeping SHA-256 for now to align with the Kyber security level. Update later if necessary].

[EDNOTE: Placeholder. ecdh-nistp256-kyber-512r3-sha256-d00@openquantumsafe.org currently follows OQS OpehSSH's method names. We will update if necessary.]

#### 2.3.2. ecdh-nistp384-kyber-768r3-sha384-d00@openquantumsafe.org

ecdh-nistp384-kyber-768r3-sha384-d00@openquantumsafe.org defines that the classical 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 Kyber512 public key and ciphertext from the 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 [EDNOTE: Add PQ KEM specification link here].

The HASH function used in the key exchange [RFC4253] is SHA-384 [nist-sha2] [RFC4634][EDNOTE: Keeping SHA-384 for now to align with the Kyber security level. Update later if necessary].

[EDNOTE: Placeholder. ecdh-nistp384-kyber-768r3-sha384-d00@openquantumsafe.org currently follows OQS OpehSSH's method names. We will update if necessary.]

#### 2.3.3. ecdh-nistp521-kyber-1024r3-sha512-d00@openquantumsafe.org

ecdh-nistp521-kyber-1024r3-sha512-d00@openquantumsafe.org defines that the classical client and server public keys C_PK1, S_PK1 belong to the NIST P-521 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-521 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-nistp521).

The post-quantum C_PK2 and S_CT2 represent Kyber512 public key and ciphertext from the 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 [EDNOTE: Add PQ KEM specification link here].

The HASH function used in the key exchange [RFC4253] is SHA-512 [nist-sha2] [RFC4634][EDNOTE: Keeping SHA-512 for now to align with the Kyber security level. Update later if necessary].

[EDNOTE: Placeholder. ecdh-nistp521-kyber-1024r3-sha512-d00@openquantumsafe.org currently follows OQS OpehSSH's method names. We will update if necessary.]

#### 2.3.4. x25519-kyber-512r3-sha256-d00@amazon.com

x25519-kyber-512r3-sha256-d00@amazon.com defines that the classical 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 Kyber512 public key and ciphertext from the 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 [EDNOTE: Add PQ KEM specification link here].

The HASH function used in the key exchange [RFC4253] is SHA-256 [nist-sha2] [RFC4634][EDNOTE: Keeping SHA-256 for now to align with the Kyber security level. Update later if necessary].

[EDNOTE: Placeholder. x25519-kyber-512r3-sha256-d00@amazon.com is experimentally following OpehSSH's experimental implementation of the sntrup4591761x25519-sha512@tinyssh.org method name, but this draft uses Kyber which was NIST's Round PQ KEM pick. We will update later if necessary.]

### 2.4. Shared Secret K

The PQ-hybrid key exchange establishes K_CL and K_PQ by using scalar multiplication and post-quantum KEM decapsulation ('Decaps') 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 the same logic as in [I-D.ietf-tls-hybrid-design] where the classical and post-quantum exchanged secrets are concatenated and used in the key schedule.

The ECDH shared secret was traditionally encoded as an integer 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. Thus, K_PQ and K_CL are encoded as fixed-length byte arrays, not as integers. Byte arrays are defined in Section 5 of [RFC4251].

[EDNOTE: The keys are derived following the same SSH logic (as explained in the Key Derivation Section)

```
Initial IV c2s: HASH(K || H || "A" || session_id)
Initial IV s2c: HASH(K || H || "B" || session_id)
Encryption key c2s: HASH(K || H || "C" || session_id)
Encryption key s2c: HASH(K || H || "D" || session_id)
Integrity key c2s: HASH(K || H || "E" || session_id)
Integrity key s2c: HASH(K || H || "F" || session_id)
```

That is option 2a key derivation.

Other key derivation options include the following SSH logic

```
(1) K = K_PQ || K_CL or
(2b) K = HMAC-HASH(K_PQ, K_CL) or
(2c) K = HMAC-HASH(0, K_PQ || K_CL)
```

Option (2a) resembles (1), but is slightly faster because the latter hashes the shared key K 6 times, so the larger the K, the more compression function invocations we will need.

This is the option implemented in OpenSSH experimentally when the sntrup4591761x25519-sha512@tinyssh.org method is used.

Or (Option 3) using the dualPRF and the Extract-and-Expand logic of TLS, NIST etc

```
K = HKDF-HASH(0, K_PQ || K_CL) // Extract
Initial IV c2s || Initial IV s2c || Encryption key c2s ||
Encryption key s2c || Integrity key c2s ||
Integrity key s2c =
      HKDF-HASH(K, H || session_id, 6(size(HASH) ) // Expand
```

Note that (2b), (2c) and (3) deviate from SSH 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 might be a separate step from this PQ-hybrid draft.

We need to decide how the keys should be derived from the PQ-ybrid shared secret K. The options that end up not being chosen should be added in an Appendix as reference. Currently we picked option 2a to follow the logic in OpenSSH with method sntrup4591761x25519-sha512@tinyssh.org, but that could change later.]

### 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-hybrid key exchange hash H is the result of computing the HASH, where HASH is the hash algorithm specified in the named PQ-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-hybrid key exchange method related packets to exceed the minimally supported packet length. This document does not define behaviour in cases where a PQ-hybrid key exchange message cause a packet to exceed the minimally supported packet length.

## 4. Acknowledgements

## 5. IANA Considerations

This memo includes requests of IANA to register new method names "ecdh-nistp256-kyber-512r3-sha256-d00@openquantumsafe.org", "ecdh-nistp384-kyber-768r3-sha384-d00@openquantumsafe.org", "ecdh-nistp521-kyber-1024r3-sha512-d00@openquantumsafe.org", and "x25519-kyber-512r3-sha256-d00@amazon.com" to be registered by IANA in the "Key Exchange Method Names" registry for SSH [IANA-SSH].

## 6. Security Considerations

[PQ-PROOF] contains proofs of security for such PQ-hybrid key exchange schemes.

[NIST-SP-800-56C] or [NIST-SP-800-135] give NIST recommendations for key derivation methods in key exchange protocols. Some PQ-hybrid combinations may combine the shared secret from a NIST-approved algorithm (e.g., ECDH using the nistp256/secp256r1 curve) with a shared secret from a non-approved algorithm (e.g., post-quantum). [NIST-SP-800-56C] lists simple concatenation as an approved method for generation of a PQ-hybrid shared secret in which one of the constituent shared secret is from an approved method. [EDNOTE: Thus, the key exchange defined here is FIPS approved assuming the ECDH exchanged parameters are FIPS approved. ]

The way the 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.

[EDNOTE: We need to decide if we want to allow variable-length secret K. RFC8731 decided not to address this potential problem due to backwards compatibility. In this spec we could do the same or say that this specification MUST only be used with algorithms which have fixed-length shared secrets (after the variant has been fixed by the algorithm identifier in the Method Names negotiation in Section 2.3. Or we could mandate variable length keys be rejected. ]

[EDNOTE: The security considerations given in [RFC5656] therefore also applies to the ECDH key exchange scheme defined in this document. Similarly for the X25519 document. PQ Algorithms are newer

and standardized by NIST. We should include text about the combination method for the KEM shared secrets. ]

[EDNOTE: Discussion on whether an IND-CCA KEM is required or whether IND-CPA suffices.] Any KEM used in the manner described in this document MUST explicitly be designed to be secure in the event that the public key is re-used, such as achieving IND-CCA2 security or having a transform like the Fujisaki-Okamoto transform [FO][HHK] applied. While it is recommended that implementations avoid reuse of KEM public keys, implementations that do reuse KEM public keys MUST ensure that the number of reuses of a KEM public key abides by any bounds in the specification of the KEM or subsequent security analyses. Implementations MUST NOT reuse randomness in the generation of KEM ciphertexts.

**Public keys, ciphertexts, and secrets should be constant length.** This document assumes that the length of each public key, ciphertext, and shared secret is fixed once the algorithm is fixed. This is the case for all NIST Round 3 finalists and alternate candidates.

# 7. References

## 7.1. Normative References


**[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>.

**[RFC4252]** Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Authentication Protocol", RFC 4252, DOI 10.17487/RFC4252, January 2006, <https://www.rfc-editor.org/info/rfc4252>.

**[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>.


## 7.2. Informative References


**[FO]** Fujisaki, E. and T. Okamoto, "Secure Integration of Asymmetric and Symmetric Encryption Schemes", DOI 10.1007/s00145-011-9114-1, Journal of Cryptology Vol. 26, pp. 80-101, December 2011, <https://doi.org/10.1007/s00145-011-9114-1>.

**[HHK]** Hofheinz, D., Hövelmanns, K., and E. Kiltz, "A Modular Analysis of the Fujisaki-Okamoto Transformation", DOI 10.1007/978-3-319-70500-2_12, Theory of Cryptography pp. 341-371, 2017, <https://doi.org/10.1007/978-3-319-70500-2_12>.

**[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-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-06, 27 February 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-hybrid-design-06>.

**[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-800-135]** 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-800-56C]** 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>.

**[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/>.

**[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>.

**[RFC4634]** Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and HMAC-SHA)", RFC 4634, DOI 10.17487/RFC4634, July 2006, <https://www.rfc-editor.org/info/rfc4634>.

**[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>.

**[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>.

**[RFC8332]** Bider, D., "Use of RSA Keys with SHA-256 and SHA-512 in the Secure Shell (SSH) Protocol", RFC 8332, DOI 10.17487/RFC8332, March 2018, <https://www.rfc-editor.org/info/rfc8332>.

**[RFC8709]** Harris, B. and L. Velvindron, "Ed25519 and Ed448 Public Key Algorithms for the Secure Shell (SSH) Protocol", RFC 8709, DOI 10.17487/RFC8709, February 2020, <https://www.rfc-editor.org/info/rfc8709>.

**[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>.

## 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