Post-quantum public key algorithms for the Secure Shell (SSH) protocol
draft-kampanakis-curdle-pq-ssh

Abstract

   This document defines 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. It
   also defines post-quantum public key authentication methods based on
   post-quantum signature schemes. These methods are defined for use in
   the SSH Authentication Protocol.

   EDNOTE: The goal of this draft is to start the standardization of PQ
   algorithms in SSH early to mitigate the potential record-and-harvest
   later with a quantum computer attacks. This draft is not expected to
   be standardized before the NIST PQ Project has standardized PQ
   algorithms. After NIST has standardized then this document will
   replace TBD1, TBD2, TBD3 with the appropriate algorithms and
   parameters before proceeding to standardization.

   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 April 2021.

Copyright Notice

Table of Contents

1. Introduction

    Secure Shell (SSH) [RFC4251] performs key establishment using key
    exchange methods based exclusively on (Elliptic Curve) Diffie-Hellman
    style schemes. SSH [RFC4252], [RFC8332], [RFC5656], [RFC8709] also
    define public key authentication methods based on RSA or ECDSA/EdDSA
    signatures schemes. The cryptographic security of these key exchange
    and signature schemes relies on certain instances of the discrete
    logarithm and integer factorization problem being computationally
    infeasable to solve for adversaries.

    However, when sufficiently large quantum computers becomes available
    these instances would no longer be computationally infeasable
    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 can record the encrypted communication
    sent between the client and server in an SSH session and then later
    decrypt the communication when sufficiently large quantum computers
    become available. This kind of attack is known as a "record-and-
    harvest" attack. "Record-and-harvest" attacks do not apply
    retroactively to authentication but a quantum computer could threaten
    SSH authentication by impersonating as a legitimate client or server.

    This document proposes to address the problem by extending the SSH
    Transport Layer Protocol [RFC4253] with hybrid key exchange methods
    and the SSH Authentication Protocol [RFC4252] with public key methods
    based on post-quantum signature schemes. A hybrid key exchange
    method maintains the same level of security provided by current key
    exchange methods, but also adds quantum resistance. The security
    provided by the individual key exchange scheme in a hybrid key
    exchange method is independent. This means that the hybrid key
    exchange method will always be at least as secure as the most secure
    key exchange scheme executed as part of the hybrid key exchange
    method.

    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.

   The corresponsonding post-quantum signature algorithms defined in the
   NIST Post-Quantum Cryptography Standardization Project [NIST_PQ] are

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

   * 'Sign(m, sk) -> sig': A deterministic signing algorithm, which
      takes as input a message 'm' and a private key 'sk' and outputs a
      signature 'sig'.

   * 'Verify(m, pk, sigma) -> pass/fail': A verification algorithm,
      which takes as input a message 'm', a public key 'pk' and a
      signature 'sig' and outputs a verification pass or failure of the
      signature on the message.

   The post-quantum KEMs used for hybrid key exchange in the document
   are TBD1, TBD2. The post-quantum signature algorithm used for key
   based authentication is TBD3. [EDNOTE: Placeholder. Algorithms will
   be identified after NIST Round 3 concludes.] The post-quantunt
   algorithms are defined in NIST Post-quantum Project
   [NIST_PQ].[EDNOTE: Update link. Algorithms can change based on
   NIST's Round 3 standardization].

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. Hybrid Key Exchange

2.1. Hybrid Key Exchange Method Abstraction

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

   The client sends

        byte SSH_MSG_HBR_INIT
        string C_INIT

   Where C_INIT would be the concatenation of C_PQ and C_CL.

   The server sends

        byte SSH_MSG_HBR_REPLY
        string S_REPLY

   Where S_REPLY would be the concatenation of S_PQ and S_CL.

   C_PQ represents the 'pk' output of the corresponding KEMs' 'KeyGen'
   at the client. S_PQ represents the ciphertext 'ct' output of the
   corresponding KEMs' 'Encaps' algorithm generated by the server to the
   client's public key. The client decapsulate ct using its private key
   which leads to K_PQ, a post-quantum shared secret for SSH.

   C_CL and S_CL represent the ephemeral public key of the client and
   server respectively used for the classical (EC)DH key exchange which
   leads to K_CL, a classical shared secret for SSH.

   XXX For Curve25519 the C_CL is represented as a fixed length 32 byte
   string, for example. XXX

2.2. Key Derivation

   The shared secrets K_CL and K_PQ are the output from the two key
   exchange schemes X and Y, respectively, that instantiates an abstract
   hybrid key exchange method Section 2. The SSH shared secret K is
   derived as the hash algorithm specified in the named hybrid key
   exchange method name over the concatenation of K_PQ and K_CL:

        K = HASH(K_PQ, K_CL)

   The resulting bytes are fed as to the key exchange method's hash
   function to generate encryption keys.

2.3. HASH

   Derivation of encryption keys MUST be done according to Section 7.2
   in [RFC4253] with a modification on the exchange hash H. The hybrid
   key exchange hash H is the result of computing the HASH, where HASH
   is the hash algorithm specified in the named 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 C_INIT, client message octet string
         string S_REPLY, server message octet string
         string K, SSH shared secret

   The HASH function used for the definitions in this specification are
   SHA-256 [nist-sha2] [RFC4634].

2.4. Hybrid Key Exchange Method Names

   The hybrid key exchange method names defined in this document are

      ecdh-nistp256-TBD1-sha256
      x25519-TBD1-sha256
      XXX Hybrid scheme currently implemented: sntrup4591761x25519-sha512@ti
nyssh.org XXX

   [EDNOTE: Placeholder. Algorithms will be identified after NIST Round
   3 concludes.]

2.4.1. ecdh-nistp256-TBD1-sha256

   ecdh-nistp256-TBD1-sha256 defines that the classical C_CL or S_CL
   from the client or server NIST P-256 curve public key as defined in
   [nist-sp800-186]. Private and public keys are generated as described
   therein. Public keys are defined as strings of 32 bytes for NIST
   P-256. The K_CL shared secret is generated from the exchanged C_CL
   and S_CL public keys as defined in [RFC5656] (key agreement method
   ecdh-sha2-nistp256) with SHA-256 [nist-sha2] [RFC4634] instead of
   SHA-256 as the hash.

   The post-quantum C_PQ or S_PQ string from the client and server are
   TBD1. The K_PQ shared secret is decapsulated from the ciphertext
   S_PQ using the client private key [EDNOTE: Placeholder. Update based
   on the algorithm identified after NIST Round 3 concludes.]

2.4.2. x25519-TBD1-sha256

x25519-TBD1-sha256 defines that the classical C_CL or S_CL from the
client or server is Curve25519 public key as defined in [RFC7748].
Private and public keys are generated as described therein. Public
keys are defined as strings of 32 bytes for Curve25519. The K_CL
shared secret is generated from the exchanged C_CL and S_CL public
keys as defined in [RFC8731] (key agreement method curve25519-sha256)
with SHA-256 [nist-sha2] [RFC4634] instead of SHA-256 as the hash.

The post-quantum C_PQ or S_PQ string from the client and server are
TBD1. The K_PQ shared secret is decapsulated from the ciphertext
S_PQ using the client private key as defined in [EDNOTE: Placeholder.
Update based on the algorithm identified after NIST Round 3
concludes.]

3. Key Authentication

[EDNOTE: Discuss if hybrid auth keys which combine classical and PQ
signatures are necessary. Since authentication cannot be broken
retroactively, even if the PQ signature algorithms got broken, we
could switch to a classical algorithm to at least keep the classical
security. On the other hand, that would take time to deploy while
these entities would be vulnerabile to impersonation attacks. Hybrid
signatures add some overhead, but could provide the peace of mind of
remaining secure with the classical algorithm without scrambling to
deploy a change even if the PQ algorithms got broken.]

3.1. Public Key Format

```
      string    "ssh-TBD3"
      string    key
```

Here, 'key' is the x-octet public key described in the TBD3
specification.

[EDNOTE: Placeholder. Algorithms will be identified after NIST Round
3 concludes.]

3.2. Signature Format

```
      string   "ssh-TBD3"
      string   signature
```

Here, 'signature' is the x-octet signature produced in accordance
with the TBD3 specification.

[EDNOTE: Placeholder. Algorithms will be identified after NIST Round 3 concludes.]

3.3. Signing and Verification

Signatures are generated according to the procedure in TBD3 specification

Signatures are verified according to the procedure in TBD3 specification

[EDNOTE: Placeholder. Algorithms will be identified after NIST Round 3 concludes.]

4. 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 named domain parameters (see Section 7) that cause any hybrid key exchange method related packets to exceed the minimally supported packet length. This document does not define behaviour in cases where a hybrid key exchange message cause a packet to exceed the minimally supported packet length.

5. Acknowledgements

6. IANA Considerations

This memo includes requests of IANA for SSH_MSG_HBR_INIT, SSH_MSG_HBR_REPLY, ecdh-nistp256-TBD1-sha256, x25519-TBD1-sha256, and ssh-TBD3.

7. Security Considerations

[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. And more. Should include something 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.

8. References

8.1. 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>.

8.2. 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., "The Transition from Classical to Post-
          Quantum Cryptography", Work in Progress, Internet-Draft,
          draft-hoffman-c2pq-07, 26 May 2020,
          <https://tools.ietf.org/html/draft-hoffman-c2pq-07>.

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

[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>.

[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
   Cisco Systems

   Email: pkampana@cisco.com

   Douglas Stebila
   University of Waterloo

   Email: dstebila@uwaterloo.ca

   Markus Friedl
   OpenSSH

   Email: markus@openbsd.org

   Torben Hansen
   AWS

   Email: htorben@amazon.com

   Dimitrios Sikeridis
   University of New Mexico

   Email: dsike@unm.edu