Secure Shell (SSH) Key Exchange Method Using Hybrid Streamlined NTRU Prime sntrup761 and X25519 with SHA-512: sntrup761x25519-sha512

Abstract

This document describes a widely deployed hybrid key exchange method in the Secure Shell (SSH) protocol that is based on Streamlined NTRU Prime sntrup761 and X25519 with SHA-512.

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 22 March 2024.

Copyright Notice

Copyright (c) 2023 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.
1. Introduction

Secure Shell (SSH) [RFC4251] is a secure remote login protocol. The key exchange protocol described in [RFC4253] supports an extensible set of methods. [RFC5656] defines how elliptic curves are integrated into this extensible SSH framework, and [RFC8731] adds curve25519-sha256 to support the pre-quantum elliptic-curve Diffie-Hellman X25519 function [RFC7748].

Streamlined NTRU Prime [NTRUPrime] [NTRUPrimePQCS] provides post-quantum small lattice-based key-encapsulation mechanisms. The variant sntrup761 instance has been implemented widely.

To hedge against attacks on either of sntrup761 or X25519 a hybrid construction may be used, with the intention that the hybrid would be secure if either of the involved algorithms are flawed.

This document describes how to implement key exchange based on a hybrid between Streamlined NTRU Prime sntrup761 and X25519 with SHA-512 [RFC6234] in SSH.

2. 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 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Key Exchange Method: sntrup761x25519-sha512

The key-agreement is done by the X25519 Diffie-Hellman protocol as described in section 3 (Key Exchange Methods) of [RFC8731], and the key encapsulation method described in [NTRUPrimePQCS].
The key exchange procedure re-use the Elliptic Curve Diffie-Hellman (ECDH) key exchange defined in section 4 (ECDH Key Exchange) and section 7.1 (ECDH Message Numbers) of [RFC5656]. The protocol flow and the SSH_MSG_KEX_ECDH_INIT and SSH_MSG_KEX_ECDH_REPLY messages are identical, except that we use different ephemeral public values Q_C and Q_S and shared secret K as described below.

The SSH_MSG_KEX_ECDH_INIT's value Q_C that holds the client's ephemeral public key MUST be constructed by concatenating the 1158 byte public key output from the key generator of sntrup761 with the 32 byte K_A = X25519(a, 9) as described in [NTRUPrimePQCS] and [RFC8731]. The Q_C value is thus 1190 bytes.

The SSH_MSG_KEX_ECDH_REPLY's value Q_S that holds the server's ephemeral public key MUST be constructed by concatenating the 1039 byte ciphertext output from the key encapsulation mechanism of sntrup761 with the 32 byte K_B = X25519(b, 9) as described in [NTRUPrimePQCS] and [RFC8731]. The Q_S value is thus 1071 bytes.

Clients and servers MUST abort if the length of the received public keys Q_C or Q_S are not the expected lengths. An abort for these purposes is defined as a disconnect (SSH_MSG_DISCONNECT) of the session and SHOULD use the SSH_DISCONNECT_KEY_EXCHANGE_FAILED reason for the message, see section 11.1 (Disconnection Message) of [RFC4253]. No further validation is required beyond what is described in [RFC7748], [RFC8731] and [NTRUPrimePQCS].

The SSH_MSG_KEX_ECDH_REPLY's signature value is computed as described in [RFC5656] with the following changes. Instead of encoding the shared secret K as 'mpint', it MUST be encoded as 'string'. The shared secret K value MUST be the 64-byte output octet string of the SHA-512 hash computed with the input as the 32-byte octet string key output from the key encapsulation mechanism of sntrup761 concatenated with the 32-byte octet string of X25519(a, X25519(b, 9)) = X25519(b, X25519(a, 9)).

4. Acknowledgements

Jan Mojzis added "sntrup4591761x25519-sha512@tinyssh.org" to TinySSH [TinySSH] in 2018 and Markus Friedl implemented it for OpenSSH [OpenSSH] during 2019. During 2020 Damien Miller replaced sntrup4591761 with sntrup761 in OpenSSH, to create "sntrup761x25519-sha512@openssh.com". TinySSH added support for it during 2021. It became the default key exchange algorithm in OpenSSH during 2022. That is identical to the "sntrup761x25519-sha512" mechanism described in this document.

This document was derived from [RFC8731].
We wish to thank the following people who contributed to this document: Roman Danyliw, Loganaden Velvindron, Panos Kampanakis, Mark Baushke.

5. Security Considerations

The security considerations of [RFC4251], [RFC5656], [RFC7748], and [RFC8731] are inherited.

Streamlined NTRU Prime sntrup761 is aiming for the standard goal of IND-CCA2 security, is widely implemented with good performance on a wide range of architectures, and has been studied by researchers for several years. However new cryptographic primitives should be introduced and trusted conservatively, and new research findings may be published at any time that may warrant implementation reconsiderations. The method described here to combine Curve25519 with sntrup761 (i.e., SHA-512 hashing the concatenated outputs) is also available for the same kind of cryptographic scrutiny.

The increase in communication size and computational requirements may be a concern for restricted computational devices, which would then not be able to take advantage of the improved security properties offered by this work.

Since sntrup761x25519-sha512 is expected to offer no reduction of security compared to curve25519-sha256, it is RECOMMENDED that it is used and preferred whenever curve25519-sha256 is used today, when the extra communication size and computational requirements are acceptable.

As discussed in the security considerations of Curve25519-sha256 [RFC8731], the X25519 shared secret K is used bignum-encoded in that document, and this raise a potential for a hash-processing time side-channel that could leak one bit of the secret due to different length of the bignum sign pad. This document resolve that problem by using string-encoding instead of bignum-encoding.

6. IANA Considerations

IANA is requested to add a new “Method Name” of “sntrup761x25519-sha512” to the "Key Exchange Method Names" registry for Secure Shell (SSH) Protocol Parameters [IANA-KEX] with a “reference” field to this RFC and the “OK to implement” field of “MAY”.

7. References

7.1. Normative References

[IANA-KEX]


7.2. Informative References


Appendix A. Test vectors
client public key sntrup761:

...01v........0
t.ko........2.a.
.M.g.f....T.He.`
|[2.....
...

|t.XF.....V[....

[...u.....]

\...

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.

\......7p.
client public key c25519:

SSH2_MSG_KEX_ECDH_REPLY

server cipher text:

client public key c25519:

SSH2_MSG_KEX_ECDH_REPLY

server cipher text:
server public key c25519:
0000: 18 6c 55 03 db 1c 38 e3 40 d7 09 24 77 46 14 b8
0016: 5e e4 7f 19 98 04 9b 90 1f f6 b9 7f b0 70 9e 32
shared secret
0000: 9b 73 7d 41 d6 cf bb 12 56 c5 8c ad 0a 6a e2 c9
0016: bf 84 a9 0a 72 91 eb 52 e4 c1 81 c8 d2 44 7b 56
client kem key:
0000: 2c 0c 5a 36 e6 77 70 b4 d8 ab 38 9a 92 96 3a cd
0016: 10 82 38 36 40 be 2d 66 08 02 b8 17 cf eb b9 be
concatenation of KEM key and ECDH shared key:
0000: 2c 0c 5a 36 e6 77 70 b4 d8 ab 38 9a 92 96 3a cd
0016: 10 82 38 36 40 be 2d 66 08 02 b8 17 cf eb b9 be
encoded shared secret:
0000: 00 00 00 40 42 54 58 44 6f 22 75 63 04 de d7 5a
0016: 1f 23 fe f9 b1 8b 36 eb e0 e6 e2 60 c3 00 12 63
0032: b0 18 3f 42 49 07 e6 d8 22 b3 b7 6c 6c 38 37 b5
0048: b4 1f b0 d0 76 35 c7 57 e6 5e fb ef cb 5b c3 8a
0064: 1a 15 a9 6d

Figure 1

Authors' Addresses

Markus Friedl
OpenSSH
Email: markus@openbsd.org

Jan Mojzis
TinySSH
Email: jan.mojzis@gmail.com

Simon Josefsson
Email: simon@josefsson.org
URI: https://blog.josefsson.org/