        Secure Shell (SSH) Key Exchange Method using Curve25519 and Curve448
                     draft-ietf-curdle-ssh-curves-07

Abstract

   This document describes the conventions for using Curve25519 and
   Curve448 key exchange methods in the Secure Shell (SSH) protocol.

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 July 5, 2018.

Table of Contents

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] describes how elliptic curves are
   integrated in SSH, and this document reuses those protocol messages.

   This document describes how to implement key exchange based on
   Curve25519 and Ed448-Goldilocks [RFC7748] in SSH. For Curve25519
   with SHA-256 [RFC6234], the algorithm we describe is equivalent to
   the privately defined algorithm "curve25519-sha256@libssh.org", which
   is currently implemented and widely deployed in libssh and OpenSSH.
   The Curve448 key exchange method is novel but similar in spirit, and
   we chose to couple it with SHA-512 [RFC6234] to further separate it
   from the Curve25519 alternative.

   This document provide Curve25519 as the preferred choice, but
   suggests that the fall back option Curve448 is implemented to provide
   an hedge against unforeseen analytical advances against Curve25519
   and SHA-256. Due to different implementation status of these two
   curves (high-quality free implementations of Curve25519 has been in
   deployed use for several years, while Curve448 implementations are
   slowly appearing), it is accepted that adoption of Curve448 will be
   slower.

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 Methods

The key exchange procedure is similar to the ECDH method described in
chapter 4 of [RFC5656], though with a different wire encoding used
for public values and the final shared secret. Public ephemeral keys
are encoded for transmission as standard SSH strings.

The protocol flow, the SSH_MSG_KEX_ECDH_INIT and
SSH_MSG_KEX_ECDH_REPLY messages, and the structure of the exchange
hash are identical to chapter 4 of [RFC5656].

The method names registered by this document are "curve25519-sha256"
and "curve448-sha512".

The methods are based on Curve25519 and Curve448 scalar
multiplication, as described in [RFC7748]. Private and public keys
are generated as described therein. Public keys are defined as
strings of 32 bytes for Curve25519 and 56 bytes for Curve448.
Clients and servers MUST fail the key exchange if the length of the
received public keys are not the expected lengths, or if the derived
shared secret only consists of zero bits. No further validation is
required beyond what is discussed in [RFC7748]. The derived shared
secret is 32 bytes when Curve25519 is used and 56 bytes when Curve448
is used. The encodings of all values are defined in [RFC7748]. The
hash used is SHA-256 for Curve25519 and SHA-512 for Curve448.

3.1. Shared Secret Encoding

The following step differs from [RFC5656], which uses a different
conversion. This is not intended to modify that text generally, but
only to be applicable to the scope of the mechanism described in this
document.

The shared secret, K, is defined in [RFC4253] and [RFC5656] as an
integer encoded as a multiple precision integer (mpint).
Curve25519/448 outputs a binary string X, which is the 32 or 56 byte

point obtained by scalar multiplication of the other side's public
key and the local private key scalar. The 32 or 56 bytes of X are
converted into K by interpreting the octets as an unsigned fixed-
length integer encoded in network byte order.

The integer K is then encoded as an mpint using the process described
in section 5 of [RFC4251] and the resulting bytes are fed as
described in [RFC4253] to the key exchange method's hash function to
generate encryption keys.

When performing the X25519 or X448 operations, the integer values
there will be encoded into byte strings by doing a fixed-length

unsigned little-endian conversion, per [RFC7748]. It is only later
when these byte strings are then passed to the ECDH code in SSH that
the bytes are re-interpreted as a fixed-length unsigned big-endian
integer value K, and then later that K value is encoded as a
variable-length signed "mpint" before being fed to the hash algorithm
used for key generation. The mpint K is then fed along with other
data to the key exchange method's hash function to generate
encryption keys.

4. Acknowledgements

The "curve25519-sha256" key exchange method is identical to the
"curve25519-sha256@libssh.org" key exchange method created by Aris
Adamantiadis and implemented in libssh and OpenSSH.

Thanks to the following people for review and comments: Denis Bider,
Damien Miller, Niels Moeller, Matt Johnston, Eric Rescorla, Ron
Frederick, Stefan Buehler.

5. Security Considerations

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

Curve25519 provide strong security and is efficient on a wide range
of architectures, and has properties that allows better
implementation properties compared to traditional elliptic curves.
Curve448 with SHA-512 is similar, but has not received the same
cryptographic review as Curve25519, and is slower, but it is provided

as an hedge to combat unforeseen analytical advances against
Curve25519 and SHA-256.

The way the derived binary secret string is encoded into a mpint
before it is hashed (i.e., adding or removing zero-bytes for
encoding) raises the potential for a side-channel attack which could
determine the length of what is hashed. This would leak the most
significant bit of the derived secret, and/or allow detection of when
the most significant bytes are zero. For backwards compatibility
reasons it was decided not to address this potential problem.

6. IANA Considerations

IANA is requested to add "curve25519-sha256" and "curve448-sha512" to
the "Key Exchange Method Names" registry for SSH [IANA-KEX] that was
created in RFC 4250 section 4.10 [RFC4250].

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>.

    [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>.

    [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>.

7.2. Informative References

   [IANA-KEX]
               Internet Assigned Numbers Authority (IANA), "Secure Shell
               (SSH) Protocol Parameters: Key Exchange Method Names",
               January 2018, <http://www.iana.org/assignments/ssh-
               parameters/ssh-parameters.xhtml#ssh-parameters-16>.

   [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>.

Appendix A. Copying conditions

   Regarding this entire document or any portion of it, the authors make
   no guarantees and are not responsible for any damage resulting from
   its use. The authors grant irrevocable permission to anyone to use,
   modify, and distribute it in any way that does not diminish the
   rights of anyone else to use, modify, and distribute it, provided
   that redistributed derivative works do not contain misleading author
   or version information. Derivative works need not be licensed under
   similar terms.

Authors' Addresses

   Aris Adamantiadis
   libssh

Email: aris@badcode.be

Simon Josefsson
SJD AB

Email: simon@josefsson.org

Mark D. Baushke
Juniper Networks, Inc.

Email: mdb@juniper.net