Stream:	Internet Engineering Task Force (IETF)		
RFC:	8731		
Category:	Standards Track		
Published:	February 2020		
ISSN:	2070-1721		
Authors:	A. Adamantiadis	S. Josefsson	M. Baushke
	libssh	*SJD AB*	*Juniper Networks, Inc.*

RFC 8731
Secure Shell (SSH) Key Exchange Method Using Curve25519 and Curve448

Abstract

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

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8731.

Copyright Notice

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] defines how elliptic curves are integrated into this extensible SSH framework, and this document reuses the Elliptic Curve Diffie-Hellman (ECDH) key exchange protocol messages defined in Section 7.1 (ECDH Message Numbers) of [RFC5656]. Other parts of [RFC5656], such as Elliptic Curve Menezes-Qu-Vanstone (ECMQV) key agreement and Elliptic Curve Digital Signature Algorithm (ECDSA), are not considered in this document.

This document describes how to implement key exchange based on Curve25519 and Curve448 [RFC7748] in SSH. For Curve25519 with SHA-256 [RFC6234][SHS], the algorithm described is equivalent to the privately defined algorithm "curve25519-sha256@libssh.org", which at the time of publication was implemented and widely deployed in libssh [libssh] and OpenSSH [OpenSSH]. The Curve448 key exchange method is similar but uses SHA-512 [RFC6234][SHS].

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 Section 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 Section 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.

The key-agreement schemes "curve25519-sha256" and "curve448-sha512" perform the Diffie-Hellman protocol using the functions X25519 and X448, respectively. Implementations **SHOULD** compute these functions using the algorithms described in [RFC7748]. When they do so, implementations **MUST** check whether the computed Diffie-Hellman shared secret is the all-zero value and abort if so, as described in Section 6 of [RFC7748]. Alternative implementations of these functions **SHOULD** abort when either the client or the server input forces the shared secret to one of a small set of values, as described in Sections 6 and 7 of [RFC7748]. Clients and servers **MUST** also abort if the length of the received public keys 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 [IANA-REASON]. No further validation is required beyond what is described in [RFC7748]. The derived shared secret is 32 bytes when "curve25519-sha256" is used and 56 bytes when "curve448-sha512" is used. The encodings of all values are defined in [RFC7748]. The hash used is SHA-256 for "curve25519-sha256" and SHA-512 for "curve448-sha512".

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 mpint K is then encoded 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 function in SSH that the bytes are reinterpreted 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. Security Considerations

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

Curve25519 with SHA-256 provides strong (~128 bits) security, is efficient on a wide range of architectures, and has characteristics that allow for better implementation properties compared to traditional elliptic curves. Curve448 with SHA-512 provides stronger (~224 bits) security with similar implementation properties; however, it has not received the same cryptographic review as Curve25519. It is also slower (larger key material and larger secure hash algorithm), but it is provided as a hedge to combat unforeseen analytical advances against Curve25519 and SHA-256 due to the larger number of security bits.

The way the derived mpint binary secret string is encoded 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.

This document provides "curve25519-sha256" as the preferred choice but suggests that the "curve448-sha512" be implemented to provide more than 128 bits of security strength should that become a requirement.

5. IANA Considerations

IANA has added "curve25519-sha256" and "curve448-sha512" to the "Key Exchange Method Names" registry for SSH [IANA-KEX] that was created in Section 4.10 of [RFC4250].

6. References

6.1. 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>.

[SHS] National Institute of Standards and Technology, "Secure Hash Standard (SHS)", FIPS PUB 180-4, DOI 10.6028/NIST.FIPS.180-4, August 2015, <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf>.

6.2. Informative References

[IANA-KEX] IANA, "Secure Shell (SSH) Protocol Parameters: Key Exchange Method Names", , <https://www.iana.org/assignments/ssh-parameters/>.

[IANA-REASON] IANA, "Secure Shell (SSH) Protocol Parameters: Disconnection Messages Reason Codes and Descriptions", , <https://www.iana.org/assignments/ssh-parameters/>.

[libssh] libssh, "The SSH Library", , <https://www.libssh.org/>.

[OpenSSH] OpenSSH group of OpenBSD, "The OpenSSH Project", , <https://www.openssh.com/>.

[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>.

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, and Stefan Buehler.

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