Stream:	Internet Engineering Task Force (IETF)
RFC:	9142
Updates:	4250, 4253, 4432, 4462
Category:	Standards Track
Published:	January 2022
ISSN:	2070-1721
Author:	M. Baushke

RFC 9142
Key Exchange (KEX) Method Updates and Recommendations for Secure Shell (SSH)

Abstract

This document updates the recommended set of key exchange methods for use in the Secure Shell (SSH) protocol to meet evolving needs for stronger security. It updates RFCs 4250, 4253, 4432, and 4462.

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/rfc9142.

Table of Contents

1. Overview and Rationale

Secure Shell (SSH) is a common protocol for secure communication on the Internet. In [RFC4253], SSH originally defined two Key Exchange (KEX) Method Names that **MUST** be implemented. Over time, what was once considered secure is no longer considered secure. The purpose of this RFC is to recommend that some published key exchanges be deprecated or disallowed as well as to recommend some that **SHOULD** and one that **MUST** be adopted.

This document updates [RFC4250], [RFC4253], [RFC4432], and [RFC4462] by changing the requirement level ("**MUST**" moving to "**SHOULD**", "**MAY**", or "**SHOULD NOT**", and "**MAY**" moving to "**MUST**", "**SHOULD**", "**SHOULD NOT**", or "**MUST NOT**") of various key exchange mechanisms. Some recommendations will be unchanged but are included for completeness.

Section 7.2 of [RFC4253] says the following:

> The key exchange produces two values: a shared secret K, and an exchange hash H. Encryption and authentication keys are derived from these. The exchange hash H from the first key exchange is additionally used as the session identifier, which is a unique identifier for this connection. It is used by authentication methods as a part of the data that is signed as a proof of possession of a private key. Once computed, the session identifier is not changed, even if keys are later re-exchanged.

The security strength of the public key exchange algorithm and the hash used in the Key Derivation Function (KDF) both impact the security of the shared secret K being used.

The hashing algorithms used by key exchange methods described in this document are: sha1, sha256, sha384, and sha512. In many cases, the hash name is explicitly appended to the public key exchange algorithm name. However, some of them are implicit and defined in the RFC that defines the key exchange algorithm name.

Various RFCs use different spellings and capitalizations for the hashing function and encryption function names. For the purpose of this document, the following are equivalent names: sha1, SHA1, and sha1; sha256, SHA256, and SHA2-256; sha384, SHA384, and SHA2-384; and sha512, SHA512, and SHA2-512.

For the purpose of this document, the following are equivalent: aes128, AES128, AES-128; aes192, AES192, and AES-192; and aes256, AES256, and AES-256.

It is good to try to match the security strength of the public key exchange algorithm with the security strength of the symmetric cipher.

There are many possible symmetric ciphers available with multiple modes. The list in Table 1 is intended as a representative sample of those that appear to be present in most SSH implementations. The security strength estimates are generally available in [RFC4086] for triple-DES and AES, as well as [NIST.SP.800-57pt1r5], Section 5.6.1.1.

Cipher Name (modes)	Estimated Security Strength
3des (cbc)	112 bits
aes128 (cbc, ctr, gcm)	128 bits
aes192 (cbc, ctr, gcm)	192 bits
aes256 (cbc, ctr, gcm)	256 bits

Table 1: Symmetric Cipher Security Strengths

The following subsections describe how to select each component of the key exchange.

1.1. Selecting an Appropriate Hashing Algorithm

The sha1 hash is in the process of being deprecated for many reasons.

There have been attacks against sha1, and it is no longer strong enough for SSH security requirements. Therefore, it is desirable to move away from using it before attacks become more serious.

The sha1 hash provides for approximately 80 bits of security strength. This means that the shared key being used has at most 80 bits of security strength, which may not be sufficient for most users.

For purposes of key exchange methods, attacks against sha1 are collision attacks that usually rely on human help rather than a pre-image attack. The sha1 hash resistance against a second pre-image is still at 160 bits, but SSH does not depend on second pre-image resistance but rather on chosen-prefix collision resistance.

Transcript Collision attacks are documented in [TRANSCRIPTION]. This paper shows that an on-path attacker does not tamper with the Diffie-Hellman values and does not know the connection keys. The attack could be used to tamper with both I_C and I_S (as defined in Section 7.3 of [RFC4253]) and might potentially be able to downgrade the negotiated ciphersuite to a weak cryptographic algorithm that the attacker knows how to break.

These attacks are still computationally very difficult to perform, but it is desirable that any key exchange using sha1 be phased out as soon as possible.

If there is a need for using sha1 in a key exchange for compatibility, it would be desirable to list it last in the preference list of key exchanges.

Use of the SHA-2 family of hashes found in [RFC6234] rather than the sha1 hash is strongly advised.

When it comes to the SHA-2 family of secure hashing functions, sha256 has 128 bits of security strength; sha384 has 192 bits of security strength; and sha512 has 256 bits of security strength. It is suggested that the minimum secure hashing function used for key exchange methods should be sha256 with 128 bits of security strength. Other hashing functions may also have the same number of bits of security strength, but none are as yet defined in any RFC for use in a KEX for SSH.

To avoid combinatorial explosion of key exchange names, newer key exchanges are generally restricted to *-sha256 and *-sha512. The exceptions are ecdh-sha2-nistp384 and gss-nistp384-sha384-*, which are defined to use sha384 for the hash algorithm.

Table 2 provides a summary of security strength for hashing functions for collision resistance. You may consult [NIST.SP.800-107r1] for more information on hash algorithm security strength.

Hash Name	Estimated Security Strength
sha1	80 bits (before attacks)
sha256	128 bits
sha384	192 bits
sha512	256 bits

Table 2: Hashing Function Security Strengths

1.2. Selecting an Appropriate Public Key Algorithm

SSH uses mathematically hard problems for doing key exchanges:

- Elliptic Curve Cryptography (ECC) has families of curves for key exchange methods for SSH. NIST prime curves with names and other curves are available using an object identifier (OID) with Elliptic Curve Diffie-Hellman (ECDH) via [RFC5656]. Curve25519 and curve448 key exchanges are used with ECDH via [RFC8731].
- Finite Field Cryptography (FFC) is used for Diffie-Hellman (DH) key exchange with "safe primes" either from a specified list found in [RFC3526] or generated dynamically via [RFC4419] as updated by [RFC8270].
- Integer Factorization Cryptography (IFC) using the RSA algorithm is provided for in [RFC4432].

It is desirable that the security strength of the key exchange be chosen to be comparable with the security strength of the other elements of the SSH handshake. Attackers can target the weakest element of the SSH handshake.

It is desirable that a minimum of 112 bits of security strength be selected to match the weakest of the symmetric cipher (3des-cbc) available. Based on implementer security needs, a stronger minimum may be desired.

The larger the Modular Exponentiation (MODP) group, the ECC curve size, or the RSA key length, the more computation power will be required to perform the key exchange.

1.2.1. Elliptic Curve Cryptography (ECC)

For ECC, across all of the named curves, the minimum security strength is approximately 128 bits. The [RFC5656] key exchanges for the named curves use a hashing function with a matching security strength. Likewise, the [RFC8731] key exchanges use a hashing function that has more security strength than the curves. The minimum strength will be the security strength of the curve. Table 3 contains a breakdown of just the ECC security strength by curve name; it does include the hashing algorithm used. The curve25519 and curve488 security-level numbers are in [RFC7748]. The nistp256, nistp384, and nistp521 (NIST prime curves) are provided in [RFC5656]. The hashing algorithm designated for use with the individual curves have approximately the same number of bits of security as the named curve.

Curve Name	Estimated Security Strength
nistp256	128 bits
nistp384	192 bits
nistp521	512 bits
curve25519	128 bits
curve448	224 bits

Table 3: ECC Security Strengths

1.2.2. Finite Field Cryptography (FFC)

For FFC, it is recommended to use a modulus with a minimum of 2048 bits (approximately 112 bits of security strength) with a hash that has at least as many bits of security as the FFC. The security strength of the FFC and the hash together will be the minimum of those two values. This is sufficient to provide a consistent security strength for the 3des-cbc cipher. Section 1 of [RFC3526] notes that the Advanced Encryption Standard (AES) cipher, which has more strength, needs stronger groups. For the 128-bit AES, we need about a 3200-bit group. The 192- and 256-bit keys would need groups that are about 8000 and 15400 bits, respectively. Table 4 provides the security strength of the MODP group. When paired with a hashing algorithm, the security strength will be the minimum of the two algorithms.

Prime Field Size	Estimated Security Strength	Example MODP Group
2048-bit	112 bits	group14
3072-bit	128 bits	group15
4096-bit	152 bits	group16

Prime Field Size	Estimated Security Strength	Example MODP Group
6144-bit	176 bits	group17
8192-bit	200 bits	group18

Table 4: FFC MODP Security Strengths

The minimum MODP group is the 2048-bit MODP group14. When used with sha1, this group provides approximately 80 bits of security. When used with sha256, this group provides approximately 112 bits of security. The 3des-cbc cipher itself provides at most 112 bits of security, so the group14-sha256 key exchanges is sufficient to keep all of the 3des-cbc key, for 112 bits of security.

A 3072-bit MODP group with sha256 hash will provide approximately 128 bits of security. This is desirable when using a cipher such as aes128 or chacha20-poly1305 that provides approximately 128 bits of security.

The 8192-bit group18 MODP group when used with sha512 provides approximately 200 bits of security, which is sufficient to protect aes192 with 192 bits of security.

1.2.3. Integer Factorization Cryptography (IFC)

The only IFC algorithm for key exchange is the RSA algorithm specified in [RFC4432]. RSA 1024-bit keys have approximately 80 bits of security strength. RSA 2048-bit keys have approximately 112 bits of security strength. It is worth noting that the IFC types of key exchange do not provide Forward Secrecy, which both FFC and ECC do provide.

In order to match the 112 bits of security strength needed for 3des-cbc, an RSA 2048-bit key matches the security strength. The use of a SHA-2 family hash with RSA 2048-bit keys has sufficient security to match the 3des-cbc symmetric cipher. The rsa1024-sha1 key exchange has approximately 80 bits of security strength and is not desirable.

Table 5 summarizes the security strengths of these key exchanges without including the hashing algorithm strength. Guidance for these strengths can be found in [NIST.SP.800-57pt1r5], Section 5.6.1.1.

Key Exchange Method	Estimated Security Strength
rsa1024-sha1	80 bits
rsa2048-sha256	112 bits

Table 5: IFC Security Strengths

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

This document adopts the style and conventions of [RFC4253] in specifying how the use of data key exchange is indicated in SSH.

This RFC also collects key exchange method names in various existing RFCs ([RFC4253], [RFC4419], [RFC4432], [RFC4462], [RFC5656], [RFC8268], [RFC8308], [RFC8731], and [RFC8732]) and provides a suggested suitability for implementation of **MUST**, **SHOULD**, **MAY**, **SHOULD NOT**, and **MUST NOT**. Any method not explicitly listed **MAY** be implemented.

Section 7.2 of [RFC4253] defines the generation of a shared secret K (really the output of the KDF) and an exchange key hash H. Each key exchange method uses a specified HASH function, which must be the same for both key exchange and Key Derivation. H is used for key exchange integrity across the SSH session as it is computed only once. It is noted at the end of Section 7.2 of [RFC4253] that "This process will lose entropy if the amount of entropy in K is larger than the internal state size of HASH", so care must be taken that the hashing algorithm used is well chosen ("reasonable") for the key exchange algorithms being used.

This document provides guidance as to what key exchange algorithms are to be considered for new or updated SSH implementations.

In general, key exchange methods that are considered "weak" are being moved to either deprecated ("**SHOULD NOT**") or disallowed ("**MUST NOT**"). Methods that are newer or considered to be stronger usually require more device resources than many administrators and/or developers need are to be allowed ("**MAY**"). (Eventually, some of these methods could be moved by consensus to "**SHOULD**" to increase interoperability and security.) Methods that are not "weak" and have implementation consensus are encouraged ("**SHOULD**"). There needs to be at least one consensus method promoted to a status of mandatory to implement (MTI). This should help to provide continued interoperability even with the loss of one of the now disallowed MTI methods.

For this document, 112 bits of security strength is the minimum. Use of either or both of sha1 and RSA 1024 bits at an approximate 80 bits of security fall below this minimum and should be deprecated and moved to disallowed as quickly as possible in configured deployments of SSH. It seems plausible that this minimum may be increased over time, so authors and administrators may wish to prepare for a switch to algorithms that provide more security strength.

3.1. Elliptic Curve Cryptography (ECC)

The Elliptic Curve (EC) key exchange algorithms used with SSH include the ECDH and EC Menezes-Qu-Vanstone (ECMQV).

The ECC curves defined for the key exchange algorithms above include the following: curve25519, curve448, the NIST prime curves (nistp256, nistp384, and nistp521), as well as other curves allowed for by Section 6 of [RFC5656]. There are key exchange mechanisms based on the Generic Security Service Application Program Interface (GSS-API) that use these curves as well that have a "gss-" prefix.

3.1.1. curve25519-sha256 and gss-curve25519-sha256-*

Curve25519 is efficient on a wide range of architectures with properties that allow higher-performance implementations compared to the patented elliptic curve parameters purchased by NIST for the general public to use as described in [RFC5656]. The corresponding key exchange methods use sha256 defined in [RFC6234]. sha2 is a reasonable hash for use in both the KDF and session integrity. It is reasonable for both gss and non-gss uses of curve25519 key exchange methods. These key exchange methods are described in [RFC8731] and [RFC8732] and are similar to the IKEv2 key agreement described in [RFC8031]. The curve25519-sha256 key exchange method has multiple implementations and **SHOULD** be implemented. The gss-curve25519-sha256-* key exchange method **SHOULD** also be implemented because it shares the same performance and security characteristics as curve25519-sha256.

Table 6 contains a summary of the recommendations for curve25519-based key exchanges.

Key Exchange Method Name	Guidance
curve25519-sha256	**SHOULD**
gss-curve25519-sha256-*	**SHOULD**

Table 6: Curve25519 Implementation Guidance

3.1.2. curve448-sha512 and gss-curve448-sha512-*

Curve448 provides more security strength than curve25519 at a higher computational and bandwidth cost. The corresponding key exchange methods use sha512 defined in [RFC6234]. SHA2-512 is a reasonable hash for use in both the KDF and session integrity. It is reasonable for both gss and non-gss uses of curve448 key exchange methods. These key exchange methods are described in [RFC8731] and [RFC8732] and are similar to the IKEv2 key agreement described in [RFC8031]. The curve448-sha512 key exchange method **MAY** be implemented. The gss-curve448-sha512-* key exchange method **MAY** also be implemented because it shares the same performance and security characteristics as curve448-sha512.

Table 7 contains a summary of the recommendations for curve448-based key exchanges.

Key Exchange Method Name	Guidance
curve448-sha512	**MAY**
gss-curve448-sha512-*	**MAY**

Table 7: Curve448 Implementation Guidance

3.1.3. ecdh-*, ecmqv-sha2, and gss-nistp*

The ecdh-sha2-* namespace allows for both the named NIST prime curves (nistp256, nistp384, and nistp521) as well as other curves to be defined for the ECDH key exchange. At the time of this writing, there are three named curves in this namespace that **SHOULD** be supported. They appear in Section 10.1 of [RFC5656]. If implemented, the named curves **SHOULD** always be enabled unless specifically disabled by local security policy. In Section 6.1 of [RFC5656], the method to name other ECDH curves using OIDs is specified. These other curves **MAY** be implemented.

The GSS-API namespace with gss-nistp*-sha* mirrors the algorithms used by ecdh-sha2-* names. They are described in [RFC8732].

ECDH reduces bandwidth of key exchanges compared to FFC DH at a similar security strength.

Table 8 lists algorithms as "**SHOULD**" where implementations may be more efficient or widely deployed. The items listed as "**MAY**" in Table 8 are potentially less efficient.

Key Exchange Method Name	Guidance
ecdh-sha2-*	**MAY**
ecdh-sha2-nistp256	**SHOULD**
gss-nistp256-sha256-*	**SHOULD**
ecdh-sha2-nistp384	**SHOULD**
gss-nistp384-sha384-*	**SHOULD**
ecdh-sha2-nistp521	**SHOULD**
gss-nistp521-sha512-*	**SHOULD**
ecmqv-sha2	**MAY**

Table 8: ECDH Implementation Guidance

It is advisable to match the Elliptic Curve Digital Signature Algorithm (ECDSA) and ECDH algorithm to use the same curve for both to maintain the same security strength in the connection.

3.2. Finite Field Cryptography (FFC)

3.2.1. FFC Diffie-Hellman Using Generated MODP Groups

[RFC4419] defines two key exchange methods that use a random selection from a set of pre-generated moduli for key exchange: the diffie-hellman-group-exchange-sha1 method and the diffie-hellman-group-exchange-sha256 method. Per [RFC8270], implementations **SHOULD** use a MODP group whose modulus size is equal to or greater than 2048 bits. MODP groups with a modulus size less than 2048 bits are weak and **MUST NOT** be used.

The diffie-hellman-group-exchange-sha1 key exchange method **SHOULD NOT** be used. This method uses sha1, which is being deprecated.

The diffie-hellman-group-exchange-sha256 key exchange method **MAY** be used. This method uses SHA-256, which is reasonable for MODP groups less than 4096 bits.

Care should be taken in the pre-generation of the moduli P and generator G such that the generator provides a Q-ordered subgroup of P. Otherwise, the parameter set may leak one bit of the shared secret.

Table 9 provides a summary of the guidance for these exchanges.

Key Exchange Method Name	Guidance
diffie-hellman-group-exchange-sha1	**SHOULD NOT**
diffie-hellman-group-exchange-sha256	**MAY**

Table 9: FFC Generated MODP Group Implementation Guidance

3.2.2. FFC Diffie-Hellman Using Named MODP Groups

The diffie-hellman-group14-sha256 key exchange method is defined in [RFC8268] and represents a key exchange that has approximately 112 bits of security strength that matches 3des-cbc symmetric cipher security strength. It is a reasonably simple transition from sha1 to SHA-2, and given that diffie-hellman-group14-sha1 and diffie-hellman-group14-sha256 share a MODP group and only differ in the hash function used for the KDF and integrity, it is a correspondingly simple transition from implementing diffie-hellman-group14-sha1 to implementing diffie-hellman-group14-sha256. Given that diffie-hellman-group14-sha1 is being removed from mandatory to implement (MTI) status, the diffie-hellman-group14-sha256 method **MUST** be implemented. The rest of the FFC MODP group from [RFC8268] have a larger number of security bits and are suitable for symmetric ciphers that also have a similar number of security bits.

Table 10 provides explicit guidance by name.

Key Exchange Method Name	Guidance
diffie-hellman-group14-sha256	**MUST**
gss-group14-sha256-*	**SHOULD**
diffie-hellman-group15-sha512	**MAY**
gss-group15-sha512-*	**MAY**
diffie-hellman-group16-sha512	**SHOULD**
gss-group16-sha512-*	**MAY**
diffie-hellman-group17-sha512	**MAY**
gss-group17-sha512-*	**MAY**
diffie-hellman-group18-sha512	**MAY**
gss-group18-sha512-*	**MAY**

Table 10: FFC Named Group Implementation Guidance

3.3. Integer Factorization Cryptography (IFC)

The rsa1024-sha1 key exchange method is defined in [RFC4432] and uses an RSA 1024-bit modulus with a sha1 hash. This key exchange does NOT meet security requirements. This method **MUST NOT** be implemented.

The rsa2048-sha256 key exchange method is defined in [RFC4432] and uses an RSA 2048-bit modulus with a sha256 hash. This key exchange meets 112-bit minimum security strength. This method **MAY** be implemented.

Table 11 provides a summary of the guidance for IFC key exchanges.

Key Exchange Method Name	Guidance
rsa1024-sha1	**MUST NOT**
rsa2048-sha256	**MAY**

Table 11: IFC Implementation Guidance

3.4. KDFs and Integrity Hashing

The sha1 and SHA-2 family of hashing algorithms are combined with the FFC, ECC, and IFC algorithms to comprise a key exchange method name.

The selected hash algorithm is used both in the KDF as well as for the integrity of the response.

All of the key exchange methods using the sha1 hashing algorithm should be deprecated and phased out due to security concerns for sha1, as documented in [RFC6194].

Unconditionally deprecating and/or disallowing sha1 everywhere will hasten the day when it may be simply removed from implementations completely. Leaving partially broken algorithms lying around is not a good thing to do.

The SHA-2 family of hashes [RFC6234] is more secure than sha1. They have been standardized for use in SSH with many of the currently defined key exchanges.

Please note that at the present time, there is no key exchange method for Secure Shell that uses the SHA-3 family of secure hashing functions or the Extendable-Output Functions ([NIST.FIPS.202]).

Prior to the changes made by this document, diffie-hellman-group1-sha1 and diffie-hellman-group14-sha1 were MTI. diffie-hellman-group14-sha1 is the stronger of the two. Group14 (a 2048-bit MODP group) is defined in [RFC3526]. The group1 MODP group with approximately 80 bits of security is too weak to be retained. However, rather than jumping from the MTI status to making it disallowed, many implementers suggested that it should transition to deprecated first and be disallowed at a later time. The group14 MODP group using a sha1 hash for the KDF is not as weak as the group1 MODP group. There are some legacy situations where it will still provide administrators with value, such as small hardware Internet of Things (IOT) devices that have insufficient compute and memory resources to use larger MODP groups before a timeout of the session occurs. There was consensus to transition from MTI to a requirement status that provides for continued use with the expectation that it would be deprecated or disallowed in the future. Therefore, it is considered reasonable to retain the diffie-hellman-group14-sha1 exchange for interoperability with legacy implementations. The diffie-hellman-group14-sha1 key exchange **MAY** be implemented, but should be put at the end of the list of negotiated key exchanges.

The diffie-hellman-group1-sha1 and diffie-hellman-group-exchange-sha1 **SHOULD NOT** be implemented. The gss-group1-sha1-*, gss-group14-sha1-*, and gss-gex-sha1-* key exchanges are already specified as **SHOULD NOT** be implemented by [RFC8732].

3.5. Secure Shell Extension Negotiation

There are two methods, ext-info-c and ext-info-s, defined in [RFC8308]. They provide a mechanism to support other Secure Shell negotiations. Being able to extend functionality is desirable. Both ext-info-c and ext-info-s **SHOULD** be implemented.

4. Summary Guidance for Implementation of Key Exchange Method Names

Table 12 provides the existing key exchange method names listed alphabetically. The Implement column contains the current recommendations of this RFC.

Key Exchange Method Name	Reference	Previous Recommendation	RFC 9142 Implement
curve25519-sha256	[RFC8731]	none	**SHOULD**
curve448-sha512	[RFC8731]	none	**MAY**
diffie-hellman-group-exchange-sha1	[RFC4419], [RFC8270]	none	**SHOULD NOT**
diffie-hellman-group-exchange-sha256	[RFC4419], [RFC8720]	none	**MAY**
diffie-hellman-group1-sha1	[RFC4253]	**MUST**	**SHOULD NOT**
diffie-hellman-group14-sha1	[RFC4253]	**MUST**	**MAY**
diffie-hellman-group14-sha256	[RFC8268]	none	**MUST**
diffie-hellman-group15-sha512	[RFC8268]	none	**MAY**
diffie-hellman-group16-sha512	[RFC8268]	none	**SHOULD**
diffie-hellman-group17-sha512	[RFC8268]	none	**MAY**
diffie-hellman-group18-sha512	[RFC8268]	none	**MAY**
ecdh-sha2-*	[RFC5656]	**MAY**	**MAY**
ecdh-sha2-nistp256	[RFC5656]	**MUST**	**SHOULD**
ecdh-sha2-nistp384	[RFC5656]	**MUST**	**SHOULD**
ecdh-sha2-nistp521	[RFC5656]	**MUST**	**SHOULD**
ecmqv-sha2	[RFC5656]	**MAY**	**MAY**
ext-info-c	[RFC8308]	**SHOULD**	**SHOULD**
ext-info-s	[RFC8308]	**SHOULD**	**SHOULD**
gss-	[RFC4462]	reserved	reserved

Key Exchange Method Name	Reference	Previous Recommendation	RFC 9142 Implement
gss-curve25519-sha256-*	[RFC8732]	**SHOULD**	**SHOULD**
gss-curve448-sha512-*	[RFC8732]	**MAY**	**MAY**
gss-gex-sha1-*	[RFC4462], [RFC8732]	**SHOULD NOT**	**SHOULD NOT**
gss-group1-sha1-*	[RFC4462], [RFC8732]	**SHOULD NOT**	**SHOULD NOT**
gss-group14-sha1-*	[RFC4462], [RFC8732]	**SHOULD NOT**	**SHOULD NOT**
gss-group14-sha256-*	[RFC8732]	**SHOULD**	**SHOULD**
gss-group15-sha512-*	[RFC8732]	**MAY**	**MAY**
gss-group16-sha512-*	[RFC8732]	**SHOULD**	**MAY**
gss-group17-sha512-*	[RFC8732]	**MAY**	**MAY**
gss-group18-sha512-*	[RFC8732]	**MAY**	**MAY**
gss-nistp256-sha256-*	[RFC8732]	**SHOULD**	**SHOULD**
gss-nistp384-sha384-*	[RFC8732]	**MAY**	**SHOULD**
gss-nistp521-sha512-*	[RFC8732]	**MAY**	**SHOULD**
rsa1024-sha1	[RFC4432]	**MAY**	**MUST NOT**
rsa2048-sha256	[RFC4432]	**MAY**	**MAY**

Table 12: IANA Guidance for Implementation of Key Exchange Method Names

The full set of official [IANA-KEX] key algorithm method names not otherwise mentioned in this document **MAY** be implemented.

5. Security Considerations

This SSH protocol provides a secure encrypted channel over an insecure network. It performs server host authentication, key exchange, encryption, and integrity checks. It also derives a unique session ID that may be used by higher-level protocols. The key exchange itself generates a shared secret and uses the hash function for both the KDF and integrity.

Full security considerations for this protocol are provided in [RFC4251] and continue to apply. In addition, the security considerations provided in [RFC4432] apply. Note that Forward Secrecy is NOT available with the rsa1024-sha1 or rsa2048-sha256 key exchanges.

It is desirable to deprecate or disallow key exchange methods that are considered weak so they are not still actively in operation when they are broken.

A key exchange method is considered weak when the security strength is insufficient to match the symmetric cipher or the algorithm has been broken.

The 1024-bit MODP group used by diffie-hellman-group1-sha1 is too small for the symmetric ciphers used in SSH.

MODP groups with a modulus size less than 2048 bits are too small for the symmetric ciphers used in SSH. If the diffie-hellman-group-exchange-sha256 or diffie-hellman-group-exchange-sha1 key exchange method is used, the modulus size of the MODP group used needs to be at least 2048 bits.

At this time, the rsa1024-sha1 key exchange is too small for the symmetric ciphers used in SSH.

The use of sha1 for use with any key exchange may not yet be completely broken, but it is time to retire all uses of this algorithm as soon as possible.

The diffie-hellman-group14-sha1 algorithm is not yet completely deprecated. This is to provide a practical transition from the MTI algorithms to a new one. However, it would be best to only be used as a last resort in key exchange negotiations. All key exchange methods using the sha1 hash are to be considered as deprecated.

6. IANA Considerations

IANA has added a new column to the "Key Exchange Method Names" registry [IANA-KEX] with the heading "OK to Implement" and annotated entries therein with the implementation guidance provided in Section 4, "Summary Guidance for Implementation of Key Exchange Method Names", in this document. IANA also added entries for ecdh-sha2-nistp256, ecdh-sha2-nistp384, and ecdh-sha2-nistp521, and added references to [RFC4462] and [RFC8732] for gss-gex-sha1-*, gss-group1-sha1-*, gss-group14-sha1-*, diffie-hellman-group-exchange-sha1, and diffie-hellman-group-exchange-sha256. A summary may be found in Table 12 in Section 4. IANA has also included this document as an additional registry reference for the suggested implementation guidance provided in Section 4 of this document and added a note indicating the following:

> OK to Implement guidance entries for registrations that pre-date [RFC9142] are found in Table 12 in Section 4 of [RFC9142].

Registry entries annotated with "**MUST NOT**" are considered disallowed. Registry entries annotated with "**SHOULD NOT**" are deprecated and may be disallowed in the future.

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>.

[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>.

[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>.

[RFC8268] Baushke, M., "More Modular Exponentiation (MODP) Diffie-Hellman (DH) Key Exchange (KEX) Groups for Secure Shell (SSH)", RFC 8268, DOI 10.17487/RFC8268, December 2017, <https://www.rfc-editor.org/info/rfc8268>.

[RFC8270] Velvindron, L. and M. Baushke, "Increase the Secure Shell Minimum Recommended Diffie-Hellman Modulus Size to 2048 Bits", RFC 8270, DOI 10.17487/RFC8270, December 2017, <https://www.rfc-editor.org/info/rfc8270>.

[RFC8308] Bider, D., "Extension Negotiation in the Secure Shell (SSH) Protocol", RFC 8308, DOI 10.17487/RFC8308, March 2018, <https://www.rfc-editor.org/info/rfc8308>.

[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>.

7.2. Informative References

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

[NIST.FIPS.202] National Institute of Standards and Technology, "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions", FIPS PUB 202, DOI 10.6028/NIST.FIPS.202, August 2015, <https://doi.org/10.6028/NIST.FIPS.202>.

[NIST.SP.800-107r1] Dang, Q., "Recommendation for applications using approved hash algorithms", DOI 10.6028/NIST.SP.800-107r1, August 2012, <https://doi.org/10.6028/NIST.SP.800-107r1>.

[NIST.SP.800-57pt1r5] Barker, E., "Recommendation for Key Management: Part 1 - General", DOI 10.6028/NIST.SP.800-57pt1r5, May 2020, <https://doi.org/10.6028/NIST.SP.800-57pt1r5>.

[RFC3526] Kivinen, T. and M. Kojo, "More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)", RFC 3526, DOI 10.17487/RFC3526, May 2003, <https://www.rfc-editor.org/info/rfc3526>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <https://www.rfc-editor.org/info/rfc4086>.

[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>.

[RFC4419] Friedl, M., Provos, N., and W. Simpson, "Diffie-Hellman Group Exchange for the Secure Shell (SSH) Transport Layer Protocol", RFC 4419, DOI 10.17487/RFC4419, March 2006, <https://www.rfc-editor.org/info/rfc4419>.

[RFC4432] Harris, B., "RSA Key Exchange for the Secure Shell (SSH) Transport Layer Protocol", RFC 4432, DOI 10.17487/RFC4432, March 2006, <https://www.rfc-editor.org/info/rfc4432>.

[RFC4462] Hutzelman, J., Salowey, J., Galbraith, J., and V. Welch, "Generic Security Service Application Program Interface (GSS-API) Authentication and Key Exchange for the Secure Shell (SSH) Protocol", RFC 4462, DOI 10.17487/RFC4462, May 2006, <https://www.rfc-editor.org/info/rfc4462>.

[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>.

[RFC6194] Polk, T., Chen, L., Turner, S., and P. Hoffman, "Security Considerations for the SHA-0 and SHA-1 Message-Digest Algorithms", RFC 6194, DOI 10.17487/RFC6194, March 2011, <https://www.rfc-editor.org/info/rfc6194>.

[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>.

[RFC8031] Nir, Y. and S. Josefsson, "Curve25519 and Curve448 for the Internet Key Exchange Protocol Version 2 (IKEv2) Key Agreement", RFC 8031, DOI 10.17487/RFC8031, December 2016, <https://www.rfc-editor.org/info/rfc8031>.

[RFC8720] Housley, R., Ed. and O. Kolkman, Ed., "Principles for Operation of Internet Assigned Numbers Authority (IANA) Registries", RFC 8720, DOI 10.17487/RFC8720, February 2020, <https://www.rfc-editor.org/info/rfc8720>.

[RFC8732] Sorce, S. and H. Kario, "Generic Security Service Application Program Interface (GSS-API) Key Exchange with SHA-2", RFC 8732, DOI 10.17487/RFC8732, February 2020, <https://www.rfc-editor.org/info/rfc8732>.

[TRANSCRIPTION] Bhargavan, K. and G. Leurent, "Transcript Collision Attacks: Breaking Authentication in TLS, IKE, and SSH", Network and Distributed System Security Symposium (NDSS), DOI 10.14722/ndss.2016.23418, February 2016, <https://doi.org/10.14722/ndss.2016.23418>.

Acknowledgements

Thanks to the following people for review and comments: Denis Bider, Peter Gutmann, Damien Miller, Niels Moeller, Matt Johnston, Iwamoto Kouichi, Simon Josefsson, Dave Dugal, Daniel Migault, Anna Johnston, Tero Kivinen, and Travis Finkenauer.

Thanks to the following people for code to implement interoperable exchanges using some of these groups as found in this document: Darren Tucker for OpenSSH and Matt Johnston for Dropbear. And thanks to Iwamoto Kouichi for information about RLogin, Tera Term (ttssh), and Poderosa implementations also adopting new Diffie-Hellman groups based on this document.

Author's Address

Mark D. Baushke
Email: mbaushke.ietf@gmail.com