Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: 4 February 2026

D. Miller
OpenSSH

3 August 2025

Host key update mechanism for SSH
draft-miller-sshm-hostkey-update-01

Abstract

This document describes an extension to allow a Secure Shell (SSH) server to inform a client of the full set of host keys it supports. This may be used for graceful host key rotation and to provide keys for additional signature algorithms to the client, supporting algorithm agility.

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 4 February 2026.

Table of Contents

1. Introduction

Secure Shell (SSH) is a cryptographic protocol for secure remote connections and login over untrusted networks. The SSH transport layer [RFC4253] includes server authentication though a public key signature. The signature algorithm used for this authentication is negotiated between the client and server at the start of the key agreement subprotocol, and the final message sent by the server at the completion of key agreement typically contains a signature made using the negotiated algorithm.

For non-certificate keys, the SSH protocol does not specify any way for a client to learn the host keys of a server. Public keys may be shared out-of-band but are more commonly learned for a given server the first time a client connects to it and trusted thereafter.

There is no facility in the SSH transport protocol that allows a server to gracefully rotate its host keys. Unless coordinated out-of-band, a server changing host keys in this model is a hard break of connection trust, as any client that had learned the previous host key would now be met with an unexpected and untrusted key attempting to authenticate the final key exchange. This situation is effectively indistinguishable from an on-path adversary hijacking the connection.

Similarly, the SSH transport protocol offers no way for a server to inform the client of keys for alternate signature algorithms that it supports; if learning keys in-band, the client must learn the key for each signature algorithm separately, and there is no way to use a previously-learned key to bootstrap trust for another.

This document describes a simple extension to the SSH protocol that provides a mechanism for a server to advertise its full set of host keys to a client, and to prove possession of the requisite private key material for each of them.

This extension takes the form of a pair of global requests (Section 4 of [RFC4254]): "hostkeys", sent by a server to advertise its set of keys and "hostkeys-prove", which may be sent by a client to request a server prove possession of the private keys corresponding to one or more of these keys.

Although this extension addresses a missing feature of the SSH transport protocol ([RFC4253]), it is implemented using a message defined for the SSH connection protocol ([RFC4254]), a notionally higher protocol layer. The reason for this is that the transport protocol lacks a defined extension message type and because some SSH protocol implementation will terminate the connection when a transport message of unsupported type is received, whereas connection protocol global requests are explicitly specified to gracefully fail when an endpoint does not support them.

1.1. 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.

2. Protocol

As mentioned above, this extension consists of a pair of global requests: "hostkeys" and "hostkeys-prove". These are both messages of type SSH_MSG_GLOBAL_REQUEST (Section 4 of [RFC4254]).

These messages may only be sent after the SSH authentication subprotocol ([RFC4252]) has has completed, as signalled by the server sending SSH_MSG_USERAUTH_SUCCESS. They MUST NOT be sent prior to the completion of user authentication.

Messages described here use the standard SSH protocol wire encoding types defined in Section 5 of [RFC4251].

2.1. hostkeys advertisment

This message is sent by a server to inform a client of the full set of public keys it supports. It is typically sent immediately after user authentication succeeds, but MAY be sent at any time later. It MUST NOT be sent more than once by a server over the course of a given SSH transport connection.

```
byte            SSH_MSG_GLOBAL_REQUEST
string          "hostkeys" or "hostkeys-00@openssh.com"
char            0 /* want reply */
string[]        hostkeys
```

Where "hostkeys" consists of one or more public key blobs encoded as described in Section 2.3. Identical host keys MUST NOT be repeated in this message, i.e. every host key in the set must be unique.

When a client receives this message, it MAY compare the set of host keys contained therein with the host keys it had recorded for the server. In cases where the client had previously recorded a host key but that key is no longer in the advertised set, it MAY delete or disable the key locally. In cases where the server advertised new, additional host keys that the client had not previously recorded, the client SHOULD request the server prove possession of the corresponding private key using the following request.

2.2. hostkeys-prove request

This message is sent by a client to request a server prove possession of the private key material corresponding to one or more public keys. It is typically sent after a client has received a "hostkeys" advertisement to request private key possession proofs only for keys it has not seen before.

```
byte            SSH_MSG_GLOBAL_REQUEST
string          "hostkeys-prove"
char            1 /* want reply */
string[]        hostkeys
```

Where "hostkeys" consists of one or more public key blobs encoded as described in Section 2.3. All hostkeys in this request MUST BE unique; no key may be repeated.

A server MUST reply to this request either with a SSH_MSG_REQUEST_FAILURE (in case of failure or unwillness to service the request) or the folloing message:

```
byte            SSH_MSG_REQUEST_SUCCESS
string[]        signatures
```

Where "signatures" consists signatures made by the requested "hostkeys". The number of signatures present MUST be identical to the number of host keys requested and they MUST appear in identical order to their counterpart host keys.

Each signature is made by signing the following structure with the corresponding key and using the signature algorithm and encoding as it is specified for SSH (e.g. Section 3.1.1 of [RFC5656] for "ecdsa-sha2-* keys). Note that RSA host keys, which unlike other key types used in SSH can support multiple signature algorithms, have additional rules described below in Section 2.2.1.

```
string          "hostkeys-prove-0"
string          session identifier
string          hostkey
```

Where the "hostkey" is the host key in question, also encoded as described in Section 2.3, and "session identifier" is the session identifier derived during the initial SSH transport key exchange as described in Section 7.2 of [RFC4253].

When a client receives a proof for new host keys, it MUST verify the signature for each, after which it MAY record the new host key as valid for the server.

2.2.1. Special handling for RSA host keys

Most key types used in SSH support only a single signature algorithm, but RSA keys are an exception in that they can generate signatures that use SHA-1, SHA-256 or SHA-512 as the signature hash. Which variant is used for host key signatures is negotiated at the start of key agreement via the algorithm names "ssh-rsa" (RSA-SHA1), "rsa-sha2-256" or "rsa-sha2-512".

If a RSA signature algorithm was selected during initial key agreement, then the same algorithm MUST be used for signing hostkey proofs for any RSA keys the server supports. An exception to this is if "ssh-rsa" (i.e. the insecure RSA-SHA1 mode) was negotiated, in which case the "hostkeys-prove" request MUST fail and the server shall return a SSH_MSG_REQUEST_FAILURE.

To avoid this situation, a server MAY skip advertising hostkeys when "ssh-rsa" is the negotiated host key signature algorithm. Alternately a server MAY exclude RSA host keys from the "hostkeys" advertisment message if other host key types are available.

If a non-RSA host key signature algorithm was selected during initial key agreement, then either of the "rsa-sha2-256" or "rsa-sha2-512" signature algorithms may be used to sign a proof for a RSA host key.

2.2.2. Legacy vendor-specific hostkey proof request

Existing implementations support the hostkey proof extension under a vendor-specific name, which appears in both the request name and in the signed data. To retain compatibility, implementations MAY support the vendor names in addition to the assigned names in this document. A hostkey proof request using the vendor-specific name will be:

```
byte            SSH_MSG_GLOBAL_REQUEST
string          "hostkeys-prove-00@openssh.com"
char            1 /* want reply */
string[]        hostkeys
```

When accepting a hostkey proof request that uses the vendor-specifc name, clients and server MUST generate and verify signatures over a structure that also uses the vendor-specific name:

```
string          "hostkeys-prove-00@openssh.com"
string          session identifier
string          hostkey
```

2.3. Public key encoding

In this extension, keys are encoded as strings using the the standard SSH wire encoding for public keys. SSH protocol key encodings are defined in [RFC4253] for "ssh-rsa" and "ssh-dss" keys, in [RFC5656] for "ecdsa-sha2-*" keys and in [RFC8709] for "ssh-ed25519" and "ssh-ed448" keys.

2.4. Advertising support for this extension

Support for this extension may be advertised by a SSH protocol server using the [RFC8308] extension mechanism using the name "hostkeys" in the SSH_MSG_EXT_INFO message.

```
    string          "hostkeys"
    string          "0" (version)
```

Note that this extension predates the existence of the [RFC8308] extension mechanism and several widely-deployed SSH implementations that support it do not advertise their ability to do so.

As global request messages are required by [RFC4254] to be gracefully ignored by an endpoint that does not support them, implementations MAY opportunistically use this extension in the absence of an [RFC8308] advertisement using the vendor-specific names mentioned above.

Likewise, clients and server MAY implement the vendor-specific names in addition to the ones described here (though note the special handling required for proof messages in Section 2.2.2). However, if a an endpoint advertises support for this extension via the EXT_INFO mechanism, the peer SHOULD use the standard global request names to invoke it instead of the vendor-specific names.

3. Applications

There are two motivating applications for this extension: graceful host key rotation and signature algorithm agility.

3.1. Host key rotation

With a degree of planning and coordination, this extension may be used to gracefully rotate a server’s host keys. To do this, a server begins by advertising via this extension one or more host keys in addition to the ones that it currently offers by default. As clients that support this extension connect to the server, these host keys will be learned and recorded as valid keys for the server. After some grace interval, or when all identified clients are known to have

connected at least once, the original host keys may be removed from the server and clients will now use the new keys by default without a discontinuity of trust.

3.2. Algorithm agility

This case is similar to host key rotation, except to note that it is possible to advertise host keys of entirely different signature algorithms. By following a process similar to the above host key rotation scheme, and assuming the SSH clients support them, it is possible to retire weak or poorly-performing signature algorithms in favour of better ones. In the near future, it is likely this mechanism will be useful to assist the deployment of Post-Quantum signature schemes without trust discontinuities.

4. IANA Considerations

This extension requires two existing registries to be modified.

4.1. Additions to SSH Connection Protocol Global Request Names

IANA is requested to insert the following entries into the table Connection Protocol Global Request Names [IANA-SSH-GLOBALREQS] under Secure Shell (SSH) Protocol Parameters [RFC4250].

Request Type	Reference
hostkeys	thisrfc, Section 2.1
hostkeys-prove-0	thisrfc, Section 2.2

Table 1

4.2. Additions to SSH Extension Names

IANA is requested to insert the following entry into the table Extension Names [IANA-SSH-EXT] under Secure Shell (SSH) Protocol Parameters [RFC4250].

Extension Name	Reference
hostkeys	thisrfc, Section 2.4

Table 2

5. Security Considerations

This extension defines a mechanism for cryptographic key management, which always needs careful handling. This section calls attention to some specific areas of concern.

5.1. Trusting keys

This mechanism cannot be used to estabish trust in a SSH server where it did not previously exist. Specifically, if a client learned a host key from an on-path adversary rather than the intended destination host, then this extension cannot improve this situation. Keys learned though this mechanism can never be more trustworthy than the key used to establish the SSH transport session.

5.2. Dangers of ignoring proof of private key possession

This host key update mechanism operates using two messages: an advertisement of host keys and a proof of private key possession. SSH implementations that implement this extension MUST implement support for both messages and MUST not record new host keys without verifying private key possession proofs.

Supporting the advertisment component alone, or recording host keys without checking their private key possession proofs allows an attack where a malicious server advertises a host key for a different legitimate server. If that host key is recorded and subsequently used by a client when connecting to the malicious server, then the malicious server could forward the connection to the legitimate server. This may result in actions intended to be performed on the malicious server being instead performed on the legitimate one.

5.3. Denial of service

This extension may require servers perform additional cryptographic signature operations, and require clients to perform additional verification operations. Implementation should enforce some limit on the maximum number of keys they will accept for both the "hostkeys" and "hostkeys-prove" messages.

5.4. Weak signature algorithms

Implementations that retain support for the weak "ssh-rsa" (RSA-SHA1) signature algorithm must pay careful attention to the considerations in Section 2.2.1. Similar special handling may be required in future if other signature algorithms are found to be weak.

5.5. Timing attack considerations

This mechanism provides a way for a SSH client to request additional signatures using the host private keys. In doing so, it magnifies the attack surface for observing timing discrepancies in the signature operation. Although a malicious client has little control over the signed data, implementations should be aware of this situation and ensure that "hostkeys-prove" replies do not reveal information useful towards recovering the private key itself.

6. Implementation Status

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [RFC7942], "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

The following example projects maintain an implementation of this protocol, all using the vendor-specific names described above.

OpenSSH OpenSSH is the originating implementation of this extension and has supported them since 2015.

Website: https://www.openssh.com/

AsyncSSH AsyncSSH is an implementation of the SSHv2 protocol on top of the Python asyncio framework. It added support for this extension in 2024.

Website: https://github.com/ronf/asyncssh

Github The SSH implementation used by Github supports this extension.

Website: https://github.com/

Russh Russsh has implemented this extension since 2023.

Website: https://github.com/Eugeny/russh

Apache Mina The Apache Mina ssh implementation has supported this extension since 2015.

Website: https://mina.apache.org/

This list is not exhaustive.

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>.

[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>.

[RFC4254] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Connection Protocol", RFC 4254, DOI 10.17487/RFC4254, January 2006, <https://www.rfc-editor.org/info/rfc4254>.

[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>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/info/rfc7942>.

[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>.

[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>.

[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>.

7.2. Informative References

[IANA-SSH-GLOBALREQS]
IANA, "Connection Protocol Global Request Names", <https://www.iana.org/assignments/ssh-parameters/>.

[IANA-SSH-EXT]
IANA, "Extension Names", <https://www.iana.org/assignments/ssh-parameters/>.

Acknowledgments

Jann Horn provided valuable feedback during the development of this extension.

Author’s Address

Damien Miller
OpenSSH
Email: djm@openssh.com
URI: https://www.openssh.com/