Workgroup:	Secure Shell Maintenance
Internet-Draft:	draft-josefsson-sshsig-format-03
Published:	7 July 2025
Intended Status:	Standards Track
Expires:	8 January 2026

Authors: S. Kinne D. Miller S. Josefsson, Ed.
OpenSSH

Lightweight Secure Shell (SSH) Signature Format

Abstract

This document describes a lightweight SSH Signature format that is compatible with SSH keys and wire formats.

Status of This Memo

Table of Contents

1. Introduction

Secure Shell (SSH) [RFC4251] is a secure remote-login protocol. It provides for an extensible variety of public key algorithms for identifying servers and users to one another.

The SSH key and signature formats have found uses outside of the interactive online SSH protocol itself. This document specify these formats.

At present, only detached and armored signatures are supported.

It is suggested that when referring to this signature format that the term "SSHSIG" is used.

2. Conventions Used In This Document

The descriptions of key and signature formats use the notation introduced in [RFC4251].

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. Armored format

The Armored SSH signatures is ASCII-encoded [RFC20] and consists of a header, a base64 encoded blob, and a footer.

The header is the string "-----BEGIN SSH SIGNATURE-----" followed by a newline. The footer is the string "-----END SSH SIGNATURE-----" immediately after a newline.

Newlines here is defined as ASCII LF. Some text transfer mechanism may use other line delimiters, however when Armored SSHSIG signatures are stored in context which are input to cryptographic hashes or otherwise subject to bit-by-bit comparisons, implementations **MUST** use ASCII LF as the line delimiter to have one canonical representation.

The header **MUST** be present at the start of every signature. Files containing the signature **MUST** start with the header. Likewise, the footer **MUST** be present at the end of every signature.

The base64 encoded blob **SHOULD** be broken up by newlines every 76 characters.

Example:

```
-----BEGIN SSH SIGNATURE-----
U1NIU0lHAAAAAQAAADMAAAALc3NoLWVkMjU1MTkAAAAgJKxoLBJBivUPNTUJUSslQTt2hD
jozKvHarKeN8uYFqgAAAADZm9vAAAAAAAAAFMAAAALc3NoLWVkMjU1MTkAAABAKNC4IEbt
Tq0Fb56xhtuE1/lK9H9RZJfON4o6hE9R4ZGFX98gy0+fFJ/1d2/RxnZky0Y7GojwrZkrHT
FgCqVWAQ==
-----END SSH SIGNATURE-----
```

4. Blob format

```
<CODE BEGINS>
#define MAGIC_PREAMBLE "SSHSIG"
#define SIG_VERSION    0x01

byte[6] MAGIC_PREAMBLE
 uint32 SIG_VERSION
 string publickey
 string namespace
 string reserved
 string hash_algorithm
 string signature

<CODE ENDS>
```

The publickey field **MUST** contain the serialisation of the public key used to make the signature using the usual SSH encoding rules, i.e [RFC4253], [RFC5656], [RFC8709], etc.

Verifiers **MUST** reject signatures with versions greater than those they support.

The purpose of the namespace value is to specify a unambiguous interpretation domain for the signature, e.g. file signing. This prevents cross-protocol attacks caused by signatures intended for one intended domain being accepted in another. The namespace value **MUST NOT** be the empty string.

The reserved value is present to encode future information (e.g. tags) into the signature. Implementations should ignore the reserved field if it is not empty.

Data to be signed is first hashed with the specified hash_algorithm. This is done to limit the amount of data presented to the signature operation, which may be of concern if the signing key is held in limited or slow hardware or on a remote ssh-agent. The supported hash algorithms for this pupose are "sha256" and "sha512". (Signature algorithms may use other hash algorithms internally.)

The signature itself is made using the SSH signature algorithm and encoding rules for the chosen key type. For RSA signatures, the signature algorithm must be "rsa-sha2-512" or "rsa-sha2-256" (i.e. not the legacy RSA-SHA1 "ssh-rsa").

This blob is encoded as a string using the [RFC4253] encoding rules and base64 encoded to form the middle part of the armored signature.

5. Signed Data, of which the signature goes into the blob above

```
<CODE BEGINS>
#define MAGIC_PREAMBLE "SSHSIG"

byte[6] MAGIC_PREAMBLE
 string namespace
 string reserved
 string hash_algorithm
 string H(message)

<CODE ENDS>
```

The preamble is the six-byte sequence "SSHSIG". It is included to ensure that manual signatures can never be confused with any message signed during SSH user or host authentication.

The reserved value is present to encode future information (e.g. tags) into the signature. Implementations should ignore the reserved field if it is not empty.

The data is concatenated and passed to the SSH signing function.

6. IANA Considerations

None

7. Security Considerations

The security considerations of all referenced specifications are inherited.

Cryptographic algorithms and parameters are usually broken or weakened over time. Implementers and users need to continously re-evaluate that cryptographic algorithms continue to provide the expected level of security.

Implementations has to follow best practices to avoid security concerns, and users needs to continously re-evaulate implementations for security vulnerabilities.

This signature format embeds the public key, which is usually already available for a verifier to perform the cryptographic verification with and to make trust decisions. When verifying a signature cryptographically, it is **RECOMMENDED** to use the locally configured public key rather than the public key provided in the signature. A bit-by-bit comparison of the public key could also be done. The public key within the signature should be treated as untrusted input, but it may be used as an identifier to find the locally trusted public key that can be used to verify the signature.

RSA public keys can be used with both SHA2-256 and RSA2-512, and implementations **MUST NOT** let library defaults chose the variant to use but instead instruct the library specifically which algorithm to use.

Some implementation do not explicitly require or validate that the namespace is not empty. We RECOMMEND that implementations reject those signatures as invalid. Comparing the namespace value before performing the signature verification **MAY** be done to provide a better error condition rather than a generic signature verification failure.

8. Acknowledgments

The text in this document is from PROTOCOL.sshsig from OpenSSH which appears to have been contributed to by at least Sebastian Kinne, Damien Miller, Markus Friedl, HARUYAMA Seigo, Pedro Martelletto, Paul Tagliamonte, Hidde Beydals, and Castedo Ellerman.

9. 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:

OpenSSH: C implementation.

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

SSHSIGLIB: Python implementation by Castedo Ellerman.

Website: https://gitlab.com/perm.pub/sshsiglib

SSHSIGN-GO: Go implementation by Benjamin Pannell at SierraSoftworks.

Website: https://github.com/SierraSoftworks/sshsign-go

SSHSIG: Go implementation by Hidde Beydals.

Website: https://github.com/hiddeco/sshsig

GO-SSHSIG: Go implementation by Paul Tagliamonte.

Website: https://github.com/paultag/go-sshsig

REKOR-PKI-SSH: Go implementation by Sigstore/Rekor.

Website: https://github.com/sigstore/rekor/tree/v1.0.1/pkg/pki/ssh

10. References

10.1. Normative References

[RFC20] Cerf, V., "ASCII format for network interchange", STD 80, RFC 20, DOI 10.17487/RFC0020, October 1969, <https://www.rfc-editor.org/rfc/rfc20>.

[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/rfc/rfc2119>.

[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/rfc/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/rfc/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/rfc/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/rfc/rfc8174>.

[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/rfc/rfc8709>.

10.2. Informative References

[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/rfc/rfc7942>.

Authors' Addresses

Sebastian Kinne
Email: skinne@google.com

Damien Miller
OpenSSH
Email: djm@mindrot.org

Simon Josefsson (editor)
Email: simon@josefsson.org