Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: 25 October 2026

D. Miller
OpenSSH

23 April 2026

SSH Certificate Format
draft-ietf-sshm-cert-00

Abstract

This document presents a simple certificate format that may be used in the context of the Secure Shell (SSH) protocol for user and host authentication.

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 25 October 2026.

Copyright Notice

Table of Contents

1. Introduction

Secure Shell (SSH) is a protocol for secure remote connections and login over untrusted networks. SSH uses public key signatures for host authentication and commonly uses them for user authentication. This document describes a lightweight certificate format for use in these contexts. It may be used for server authentication as part of key exchange (Section 7 of [RFC4253]) and for user public key authentication (Section 7 of [RFC4252]).

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. Certificate Format

All values in a certificate are encoded using the SSH wire representations specified by Section 5 of [RFC4251]. Additionally, the type "byte[]" indicates a sequence of bytes where the length is determined by context. All lengths are in bytes unless otherwise specified.

2.1. Public certificate format

2.1.1. General structure and common fields

Certificates have the following general structure:

string	key type
string	nonce
byte[]	public key fields
uint64	serial number
uint32	certificate role
string	identifier
string	principals
uint64	valid after
uint64	valid before
string	critical options
string	extensions
string	reserved
string	signature key
string	signature

The "key type" string identifies the certificate type, e.g. "ssh-ed25519-cert" identifies an ED25519 certificate. The valid certificate types are listed below.

The "nonce" field contains a random nonce. This value MUST be at least 16 bytes in length. 32 bytes is typical.

The "public key fields" section contains the actual public key material. These are specified below for each certificate type.

The "certificate role" field indicates the role of the certificate, either SSH2_CERT_TYPE_USER for user authentication certificates or SSH2_CERT_TYPE_HOST authentication certificates.

The "identifier" field contains a UTF-8 encoded, human-readable identifier for the certificate, suitable for display to the user, recording in logs, etc.

The "principals" field contains one or more UTF-8 encoded names, themselves encoded as strings, for example:

```
    string              principal[0]
    string              principal[1]
    ...
    string              principal[N]
```

These names identify the principals the certificate is intended to authenticate. In the case of user certificates, these will typically be user names, whereas for host certificates they will be host names and/or numerical addresses.

The "valid after" and "valid before" field represent a validity interval for the certificate with each field being interpreted as a number of seconds since the Unix epoch (00:00:00 UTC on 1 January 1970) except for two special values. A zero value in the "valid after" field indicates that the certificate is valid from any time to the "valid before" date. A all-1s (i.e. 0xFFFFFFFFFFFFFFFF) value in the "valid before" field indicates that the certificate has no end expiry date.

The "critical options" field contains zero or more of the certificate options described in Section 2.4 and encoded as described in Section 2.2. If an implementation does not recognise or support a particular option contained in a certificate, then it MUST refuse to accept the certificate for authentication.

The "extensions" field similarly contains zero or more of the certificate extensions described in Section 2.3 and encoded as described in Section 2.2. An implementation that does not recognise or support an extension present in a certificate extension section MUST ignore the extension.

The "reserved" field is reserved for future use. Implementations MUST ignore the contents of this field if it is not empty.

The "signature key" field contains the certification authority (CA) key used to sign the certificate, encoded as a SSH public key blob. Implementations MUST NOT accept certificate keys as CA keys.

The "signature" field contains a signature, made using the CA signature key over the entire certificate excluding the "signature" field itself (i.e. everything from the "key type" up to and including the "signature key"). The signature is encoded using the standard SSH signature encoding for the CA key type in question, e.g. Section 3.1.2 of [RFC5656] for ECDSA CA keys.

2.1.2. DSA certificates

DSA certificates have key type "ssh-dss-cert" and certify DSA keys as defined in Section 6.6 of [RFC4253]. This format is equivalent to the vendor extension "ssh-dss-cert-v01@openssh.com".

```
    string              "ssh-dss-cert"
    string              nonce
    mpint               p
    mpint               q
    mpint               g
    mpint               y
    uint64              serial number
    uint32              certificate role
    string              identifier
    string              principals
    uint64              valid after
    uint64              valid before
    string              critical options
    string              extensions
    string              reserved
    string              signature key
    string              signature
```

The "p", "q", "g" values are the DSA domain parameters. "y" is the public key value. These values are as defined by Section 4.1 of [FIPS.186-4].

2.1.3. ECDSA certificates

ECDSA certificates are represented by the key types "ecdsa-sha2-nistp256-cert", "ecdsa-sha2-nistp384-cert" and "ecdsa-sha2-nistp521-cert". They respectively certify "ecdsa-sha2-nistp256", "ecdsa-sha2-nistp384" and "ecdsa-sha2-nistp521" respectively, as defined by Section 3.1 of [RFC5656]. This format is equivalent to the vendor extensions "ecdsa-sha2-nistp256-cert-v01@openssh.com", "ecdsa-sha2-nistp384-cert-v01@openssh.com" and "ecdsa-sha2-nistp521-cert-v01@openssh.com".

```
    string              "ecdsa-sha2-nistp256-cert" |
                        "ecdsa-sha2-nistp384-cert" |
                        "ecdsa-sha2-nistp521-cert"
    string              nonce
    string              ecdsa_curve_name
    string              Q
    uint64              serial number
    uint32              certificate role
    string              identifier
    string              principals
    uint64              valid after
    uint64              valid before
    string              critical options
    string              extensions
    string              reserved
    string              signature key
    string              signature
```

The value "Q" is the ECDSA public value as defined by Section 6.2 of [FIPS.186-5].

2.1.4. EDDSA certificates

EDDSA certificates have key type "ssh-ed25519-cert" or "ssh-ed448-cert" and certify Ed25519 and Ed448 keys (respectively) as defined by [RFC8709]. This format is equivalent to the vendor extension "ssh-ed25519-cert-v01@openssh.com".

```
    string              "ssh-ed25519-cert" | "ssh-ed448-cert"
    string              nonce
    string              ENC(A)
    uint64              serial number
    uint32              certificate role
    string              identifier
    string              principals
    uint64              valid after
    uint64              valid before
    string              critical options
    string              extensions
    string              reserved
    string              signature key
    string              signature
```

The value ENC(A) is the EDDSA public key, the contents and interpretation of which are defined by Section 3.2 of [RFC8032].

2.1.5. RSA certificates

RSA certificates have type "ssh-rsa-cert". These certify RSA, as defined in Section 6.6 of [RFC4253]. This format is equivalent to the vendor extension "ssh-rsa-cert-v01@openssh.com".

```
    string              "ssh-rsa-cert"
    string              nonce
    mpint               e
    mpint               n
    uint64              serial number
    uint32              certificate role
    string              identifier
    string              principals
    uint64              valid after
    uint64              valid before
    string              critical options
    string              extensions
    string              reserved
    string              signature key
    string              signature
```

"n" is the public composite modulus and "e" is the public exponent. These values are defined by Section 5.1 of [FIPS.186-4].

2.1.5.1. RSA SHA2 signature algorithms

[RFC8332] specifies RSA variants that use SHA256 and SHA512. Advertising support for these algorithms with certificates (e.g. in the SSH_MSG_KEXINIT server_host_key_algorithms field) is possible using the algorithm names "rsa-sha2-256-cert" and "rsa-sha2-512-cert". These names should only be used when advertising or requesting signature algorithms and should never appear as the key type in serialised public keys.

The equivalent pre-standardisation vendor extension names are "rsa-sha2-256-cert-v01@openssh.com" and "rsa-sha2-512-cert-v01@openssh.com".

2.1.6. Other certificate formats

SSH clients and server clients MAY support additional key types not documented here. Vendor-specific key types should use the domain-qualified naming convention defined in Section 4.2 of [RFC4251].

2.2. Option / extension encoding

The "critical options" and "extensions" certificate sections use the same format for encoding, though they differ in how unsupported options are handled.

Both sections consist of zero or more key/value pairs, encoded as a pair of strings:

```
string              key
string              value
```

"key" should be a UTF-8 encoded string that identifies the option being encoded. Available keys for extensions and for critical options sections are listed below, but implementations may add their own options using the domain-qualified naming convention defined in Section 4.2 of [RFC4251].

The contents and format of "value" are specific to each option or extension. Key/value pairs MUST be ordered lexically by key name.

A common pattern for flag-type options is for the value to be the empty string. For example, this is a complete extensions section that encodes a single flag value "permit-user-rc".

```
  00000016                          # Extensions section len=22
    0000000e                        # 1st key string len=14
      7065726d69742d757365722d7263   #   "permit-user-rc"
    00000000                        # 1st value string len=0
```

For options whose values encode textual values, it is common for the value to contain a nested string. For example, the following is a complete critical options section that sets a string-values option "force-command" to the value "sftp".

```
  0000001d                          # Criticial options len=29
    0000000d                        # 1st key string len=13
      666f7263652d636f6d6d616e64    #   "force-command"
    00000008                        # 1st value string len=8
      00000004                      #   value body string len=4
        73667470                    #     "sftp"
```

Finally, the following is an example of a critical options section that contains a flag option "foo@example.com" and a string-valued option "force-command" with a the value "sftp".

```
  00000038                          # Criticial options len=56
    0000000f                        # 1st key string len=15
      666f6f406578616d706c652e636f6d #   "foo@example.com"
    00000000                        # 1st value string len=0
    0000000d                        # 2nd key string len=13
      666f7263652d636f6d6d616e64    #   "force-command"
    00000008                        # 2nd value string len=8
      00000004                      #   value body string len=4
        73667470                    #     "sftp"
```

2.3. Certificate extensions

Certificate extensions are encoded using the rules in Section 2.2. When an implementation encounters an extension it does not recognise or support, it MUST ignore it and continue processing the certificate without error.

There are no defined certificate extensions for host certificates.

The initially defined certificate extensions for user certificates are:

no-touch-required Valid for key types that support user-presence

assertions in their signatures (such as FIDO-backed keys). This option indicates that signatures made with this certificate that do not assert user-presence should be accepted, reversing the default behaviour of refusing such signatures. Note that no such key types are currently standardised, but some SSH implementation support them as vendor extensions.

This is a flag-type option. Its value is the empty string.

permit-agent-forwarding Indicates that sessions authenticated with this certificate may request authentication agent forwarding [I-D.ietf-sshm-ssh-agent]. Certificates that lack this extension MUST not permit this protocol feature be enabled on SSH server implementations that support it.

This is a flag-type option. Its value is the empty string.

permit-port-forwarding Indicates that sessions authenticated with this certificate may request authentication TCP forwarding using the "tcpip-forward" and/or "direct-tcpip" SSH channel requests defined in Section 7 of [RFC4254]. Certificates that lack this extension MUST not permit these protocol features be enabled on SSH server implementations that support them.

This is a flag-type option. Its value is the empty string.

permit-pty Indicates that sessions authenticated with this certificate may request pseudo-terminal allocation using the "pty-req" SSH channel request defined in Section 6.2 of [RFC4254]. Certificates that lack this extension MUST not permit this protocol feature be enabled on SSH server implementations that support it.

This is a flag-type option. Its value is the empty string.

permit-user-rc Indicates that sessions authenticated with this certificate may execute optional SSH-specific user initialisation scripts (e.g. "~/.ssh/rc"). Certificates that lack this extension MUST not process these scripts on SSH server implementations that support them.

This is a flag-type option. Its value is the empty string.

permit-X11-forwarding Indicates that sessions authenticated with

this certificate may request X11 protocol forwarding using the "x11-req" SSH channel request defined in Section 6.3 of [RFC4254]. Certificates that lack this extension MUST not permit this protocol feature be enabled on SSH server implementations that support it.

This is a flag-type option. Its value is the empty string.

SSH implementations may define additional vendor extensions using the domain-qualified naming convention defined in Section 4.2 of [RFC4251].

2.4. Critical options

Certificate critical options are encoded using the rules in Section 2.2. When an implementation encounters an option it does not recognise or support, it MUST refuse to accept the certificate for authorisation decisions.

There are no defined critical options for host certificates.

The initially defined critical options for user certificates are:

force-command Forces the execution of the specified command for any session (Section 6.5 of [RFC4254]) of type "exec" or "shell" and overrides the subsystem name for sessions of type "subsystem".

This is a string-type option. Its value contains a nested string which holds the command string.

source-address Provides an allow-list of source addresses from which the certificate is valid. The address list consists of zero or more CIDR ranges (e.g. "192.0.2.0/29") or wildcard addresses (e.g. "192.0.2.*") separated by commas. If this option is present, servers MUST refuse connections from sources that are not listed.

This is a string-type option. Its value contains a nested string which holds the allow-list.

verify-required Valid for key types that support user verification assertions in their signatures (such as FIDO-backed keys). This option indicates that signatures made with this certificate that must require this assertion to be present in the signature and that servers MUST refuse authentication if it is absent. Note that no such key types are currently standardised, but some SSH implementation support them as vendor extensions.

This is a flag-type option. Its value is the empty string.

SSH implementations may define additional vendor extensions using the domain-qualified naming convention defined in Section 4.2 of [RFC4251].

2.5. Private certificate format

For cases where a certificate and its associated private key must be serialised, such as adding one to a SSH authentication agent [I-D.ietf-sshm-ssh-agent], the following format is used;

```
    string              key type
    string              certificate blob
    byte[]              private key fields
```

Where "key type" is the type of the certificate being encoded (e.g. "ssh-rsa-cert"), "certificate blob" is the entire encoded certificate as described in Section 2.1 and "private key fields" contain the encoded private key values as described in the following sections.

2.5.1. DSA certificates

DSA certificates with private keys use the following encoding format.

```
    string              "ssh-dss-cert"
    string              certificate blob
    mpint               x
```

The "x" parameter is defined as per Section 4.1 of [FIPS.186-4].

2.5.2. ECDSA certificates

ECDSA certificates with private keys use the following encoding format.

```
    string              "ecdsa-sha2-nistp256-cert" |
                        "ecdsa-sha2-nistp384-cert" |
                        "ecdsa-sha2-nistp521-cert"
    string              certificate blob
    mpint               d
```

The "d" value is defined by Section 6.2 of [FIPS.186-5].

2.5.3. EDDSA certificates

EDDSA certificates with private keys use the following encoding format.

```
    string              "ssh-ed25519-cert" | "ssh-ed448-cert"
    string              certificate blob
    string              k || ENC(A)
```

The encoded value is a concatenation of the private key k and the public ENC(A) key. The contents and interpretation of the ENC(A) and k values are defined by Section 3.2 of [RFC8032].

2.5.4. RSA certificates

RSA certificates with private keys use the following encoding format.

```
    string              "ssh-rsa-cert"
    string              certificate blob
    mpint               d
    mpint               iqmp
    mpint               p
    mpint               q
```

"p" and "q" are its constituent private prime factors. "iqmp" is the inverse of "q" modulo "p". All these values except "iqmp" (which can be calculated from the others) are defined by Section 5.1 of [FIPS.186-4].

3. Using certificates

Certificates may be appear and be used any place in the SSH protocol where plain keys are used. In particular, they may be used for server authentication or for user public-key authentication. For each of these cases, the certificate, encoded as per Section 2.1 appears in place of the encoded key.

3.1. Accepting certificates

Implementations that accept a certificate MUST verify the CA signature over the certificate contents prior to making any authentication or authorisation decisions. Implementations MAY elect to verify the certificate signature as soon as the certificate is received.

When making authentication or authorisation decisions with a certificate, implementations:

* MUST check that the certificate is well-formed

* MUST ensure that no unsupported critical options are present

* MUST check that the "certificate role" matches the intended decision type (user or host authentication)

* MUST check the time against the certificate validity interval

* MUST verify that at least one of the listed certificate principals is accepted

* MUST perform any additional authorisation operations required by critical options.

3.2. Certificate signatures

When signatures are required, they are made and formatted using the same rules for the underlying plain key type. For example, a "ssh-rsa-cert" could produce signatures of types "ssh-rsa" using the rules from Section 6.6 of [RFC4253], or "rsa-sha2-256" or "rsa-sha2-512" using the rules from Section 3 of [RFC8332].

3.3. Use in host authentication

Certificate host keys may be used for server authentication as part of the initial key exchange or for subsequent re-exchange. To use a certificate host key, its key type name (e.g. "ssh-ed25519-cert") must appear in the server_host_key_algorithms of the SSH_MSG_KEXINIT (Section 7.1 of [RFC5656]). If negotiated by both the server and the client, then a certificate of the agreed type may take the place of the plain host key in the final reply message. For example in ECDH host authentication (Section 4 of [RFC5656]), host authentication occurs in the final SSH_MSG_KEX_ECDH_REPLY message:

```
byte                SSH_MSG_KEX_ECDH_REPLY
string              K_S, server's certificate
string              Q_S, server's ephemeral ECDH public key
string              the signature on the exchange hash
```

Certificates used for host authentication MUST have "certificate role" of SSH2_CERT_TYPE_HOST. Other certificate types MUST NOT be accepted.

Certificate host keys list hostnames and/or IP addresses for the host in their "principals" section. Clients MUST match the hostname or address thay are using for the host against the principals in this list and refuse to authorise the certificate if it is absent.

Clients that accept certified host keys MAY downgrade the certificate to a plain public key and process it accordingly, if they are unable to verify the certificate. This allows recovery from the situation

where the endpoints negotiate use of a cerificate host key but the client is not able to authorise the certificate contents, perhaps because the CA is not trusted or because the host name the client is using does not match any in the certificate.

3.4. Use in user authentication

For user public key authentication No prior negotiation is needed to send a user authentication attempt that contains a certificate, though clients may wish to filter the ones they send against the server's list of accepted publickey authentication signature algorithms if one was provided via the [RFC8308] SSH_MSG_EXT_INFO mechanism.

Certificates used for user authentication MUST have "certificate role" of SSH2_CERT_TYPE_USER. Other certificate types MUST not be accepted.

Certificate keys for user authentication list one or more user principal names in their "principals" section. Interpretation and authorisation of principal names for a given target user name is fully determined by the server. In the trivial case, a certificate may list user names directly and the server merely matches the user name in the SSH_MSG_USERAUTH_REQUEST (Section 7 of [RFC4252]) to one of the principals in the certificate.

To make a user authentication attempt using a certificate user key, the certificate takes the place of the plain public key in a SSH_MSG_USERAUTH_REQUEST message. For example:

byte	SSH_MSG_USERAUTH_REQUEST
string	user name
string	service name
string	"publickey"
boolean	TRUE
string	certificate key algorithm name
string	encoded certificate
string	signature

4. Protocol numbers

4.1. Certificate types

The following numbers are used for certificate types:

SSH2_CERT_TYPE_USER	1
SSH2_CERT_TYPE_HOST	2

5. IANA Considerations

This format requires creation of one new registry and additions to an existing registry.

5.1. Additions to SSH key type registry

IANA is requested to insert the following entries into the table Public Key Algorithm Names [IANA-SSH-KEYTYPES] under Secure Shell (SSH) Protocol Parameters registry group [RFC4250].

Key type name	Reference
ssh-dss-cert	Section 2.1.2
ecdsa-sha2-nistp256-cert	Section 2.1.3
ecdsa-sha2-nistp384-cert	Section 2.1.3
ecdsa-sha2-nistp521-cert	Section 2.1.3
ssh-ed25519-cert	Section 2.1.4
ssh-ed448-cert	Section 2.1.4
ssh-rsa-cert	Section 2.1.5
rsa-sha2-256-cert	Section 2.1.5.1
rsa-sha2-512-cert	Section 2.1.5.1

Table 1

5.2. New registry: Certificate Extension Identifiers

This registry, titled "Certificate Extension Identifiers" records identifiers for certificate extensions. It should be created in the Secure Shell (SSH) Protocol Parameters registry group [RFC4250]. Its initial state should consist of the following names. Future name allocations shall occur via EXPERT REVIEW as per [RFC8126].

Extension name	Reference
no-touch-required	Section 2.3
permit-agent-forwarding	Section 2.3
permit-port-forwarding	Section 2.3
permit-pty	Section 2.3
permit-user-rc	Section 2.3
permit-X11-forwarding	Section 2.3

Table 2

5.3. New registry: Certificate Criticial Option Identifiers

This registry, titled "Certificate Critical Option Identifiers" records identifiers for certificate critical options. It should be created in the Secure Shell (SSH) Protocol Parameters registry group [RFC4250]. Its initial state should consist of the following names. Future names allocations shall occur via EXPERT REVIEW as per [RFC8126].

Option name	Reference
force-command	Section 2.4
source-address	Section 2.4
verify-required	Section 2.4

Table 3

5.4. New registry: Certificate Type Numbers

This registry, titled "Certificate Type Numbers" records the numbers used to indicate the types of certificates. It should be created in the Secure Shell (SSH) Protocol Parameters registry group [RFC4250]. Its initial state should consist of the following numbers. Future message number allocations shall occur via EXPERT REVIEW as per [RFC8126].

Number	Identifier	Reference
1	SSH2_CERT_TYPE_USER	Section 4.1
2	SSH2_CERT_TYPE_HOST	Section 4.1

Table 4

6. Security Considerations

The security of any public key certificate system depends ultimately on the certification authority (CA). If the certification authority can be pursuaded by an attacker to sign certificates that are not intended, or if the private key material for the CA is leaked to an attacker, then the attacker gains the ability to create improper certificates that will be accepted by all endpoints that trust the CA. Specifying the exact process by which certificates are issued and the operational requirements of running a CA is beyond the scope of this document, but certification authorities MUST ensure that signing requests are authentic and that they are authorised by the CA's policy. Similarly, CAs MUST protect their private key material against theft, accidental disclosure or leakage. Such protection may be afforded by isolating the CA private key in a separate signing service, storing it in a hardware token or security module and/or by keeping it offline except when it is explicitly needed. Certification authorities MUST select their signing key algorithm and strength to be sufficiently secure for the expected operational life of the signing key.

For completeness, this document specifies certificate formats that are peers for all currently-specified SSH key and signature types, but not all SSH key and signature types are currently recommended for use. Specifically, the DSA key type (and associated certificate format) SHOULD NOT be used as the algorithm, as instantiated in SSH, is unacceptable weak as it uses a fixed 1024 bit key length, which is insufficient, and requires the use of SHA1, which is known to be broken, as a signature hash. Similarly the original "ssh-rsa" signature type also uses SHA1 as a signature hash and so SHOULD NOT be used. SSH clients, servers and other tools processing SSH certificates SHOULD implement allowlists of accepted key and signature types and SHOULD by default disallow the ssh-dss-cert certificate type and certificate signatures made using the ssh-rsa signature algorithm.

This specification does not define a revocation mechanism for certificates that should no longer be accepted, for example if their private key material is known to have been compromised or if it is no longer desirable to accept the principal they identify for policy or operational reasons. Implementations that accept SSH certificates SHOULD have some mechanism to disallow individual certificates that are no longer acceptable despite their being otherwise valid. In its simplest form, this mechanism could be a simple list of certificate serial numbers to disallow, or an explicit list of certificates, encoded as serialised public keys using [RFC4716] or another public key encoding.

This document does not specify how SSH clients or servers come to trust CA keys. It is RECOMMENDED that implementations define a clear mechanism to define trust of a CA public key and its desired scope. For example, a SSH client SHOULD allow trusting a CA key for only a defined set of domain names such as subdomains of an organisation’s TLD. Any limits that a client or server places on the scope of a trusted CA key MUST compose sensibly and safely with restrictions specified within certificates issued from that CA. For example, if a SSH server supports trusting a CA key but selectively disallows SSH TCP/IP port forwarding (Section 7 of [RFC4254]), then the "permit-port-forwarding" critical option would not be expected to re-enable it.

7. 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 implementations of SSH certificates. All have implemented them in the vendor extension namespace, e.g. "ssh-rsa-cert-v01@openssh.com".

OpenSSH OpenSSH is the originating implementation of this protocol and has supported it since 2010.

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

libssh libssh has supported this format since 2011.

Website: https://www.libssh.org/

Golang x/crypto/ssh The Go programming language project has supported an implementation of this format in its external "x" repository since 2012.

Website: https://pkg.go.dev/golang.org/x/crypto/ssh

Net::SSH The Net::SSH library has supported this format since 2014.

Website: https://net-ssh.github.io/net-ssh/

AsyncSSH The AsyncSSH library has supported this format since 2015.

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

Paramiko The Paramiko library has supported this format since 2017

Website: https://www.paramiko.org/

Apache MINA SSHD The Apache MINA SSHD server library has supported this format since 2020.

Website: https://mina.apache.org/sshd-project/

PuTTY The PuTTY SSH client has supported this format since 2022.

Website: https://www.chiark.greenend.org.uk/~sgtatham/putty/

russh The russh library has supported this format since 2024.

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

Additionally, Google’s HIBA system [HIBA] builds on top of the OpenSSH certificate format to perform centralised authorisation for pool of target hosts.

8. References

8.1. 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>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/info/rfc8032>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[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>.

[RFC8332] Bider, D., "Use of RSA Keys with SHA-256 and SHA-512 in the Secure Shell (SSH) Protocol", RFC 8332, DOI 10.17487/RFC8332, March 2018, <https://www.rfc-editor.org/info/rfc8332>.

[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>.

[I-D.ietf-sshm-ssh-agent] Miller, D., "SSH Agent Protocol", Work in Progress, Internet-Draft, draft-ietf-sshm-ssh-agent-02, 16 March 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-sshm-ssh-agent-02>.

[FIPS.186-4] National Institute of Standards and Technology, "Digital Signature Standard (DSS)", FIPS PUB 186-4, DOI 10.6028/NIST.FIPS.186-4, July 2013, <https://doi.org/10.6028/NIST.FIPS.186-4>.

[FIPS.186-5] National Institute of Standards and Technology, "Digital Signature Standard (DSS)", FIPS PUB 186-5, DOI 10.6028/NIST.FIPS.186-5, February 2023, <https://doi.org/10.6028/NIST.FIPS.186-5>.

8.2. Informative References

[RFC4716] Galbraith, J. and R. Thayer, "The Secure Shell (SSH) Public Key File Format", RFC 4716, DOI 10.17487/RFC4716, November 2006, <https://www.rfc-editor.org/info/rfc4716>.

[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>.

[IANA-SSH-KEYTYPES] IANA, "Public Key Algorithm Names", <https://www.iana.org/assignments/ssh-parameters/>.

[HIBA] Google, "HIBA: Host Identity Based Authorization", <https://github.com/google/hiba>.

Acknowledgements

Jeremy Norris and Yiyue Wang offered valuable feedback on this document.

Example certificates

This is an example of a hex-encoded certificate for an ecdsa-sha2-nistp256-cert public key, signed by a ssh-ed25519 certificate authority.

```
0000: 0000 0018 6563 6473 612d 7368 6132 2d6e  ....ecdsa-sha2-n
0010: 6973 7470 3235 362d 6365 7274 0000 0020  istp256-cert...
0020: 7ee0 cb87 8240 788b 087e 0a23 f505 1828  ~....@x..~.#...(
0030: 98e1 510f b3a2 fcf6 4086 30f6 25b1 aa19  ..Q.....@.0.%...
0040: 0000 0008 6e69 7374 7032 3536 0000 0041  ....nistp256...A
0050: 04a5 7c7c 0829 b7ee 3473 1383 52b8 afc1  ..||.)..4s..R...
0060: 673c 7145 29de 5f0f a699 4f08 6f58 8f96  g<qE)._...O.oX..
0070: c9ad 70d3 0b20 a4e9 3558 ffd3 f85d ddfa  ..p.. ..5X...]..
0080: 313d 294b 2b9f 2862 dd2b 5831 3688 065c  1=)K+.(b.+X16..\
0090: 52ab 54a9 8ceb 1f0a d200 0000 0100 0000  R.T.............
00a0: 136a 6f73 6566 2e6b 4065 7861 6d70 6c65  .josef.k@example
00b0: 2e6f 7267 0000 001e 0000 0007 6a6f 7365  .org........jose
00c0: 662e 6b00 0000 0f45 5841 4d50 4c45 5c6a  f.k....EXAMPLE\j
00d0: 6f73 6566 2e6b 0000 0000 4d4a 2972 0000  osef.k....MJ)r..
00e0: 0000 82e9 0790 0000 0020 0000 000d 666f  ......... ....fo
00f0: 7263 652d 636f 6d6d 616e 6400 0000 0b00  rce-command.....
0100: 0000 0765 7865 6375 7465 0000 0082 0000  ...execute......
0110: 0015 7065 726d 6974 2d58 3131 2d66 6f72  ..permit-X11-for
0120: 7761 7264 696e 6700 0000 0000 0000 1770  warding........p
0130: 6572 6d69 742d 6167 656e 742d 666f 7277  ermit-agent-forw
0140: 6172 6469 6e67 0000 0000 0000 0016 7065  arding........pe
0150: 726d 6974 2d70 6f72 742d 666f 7277 6172  rmit-port-forwar
0160: 6469 6e67 0000 0000 0000 000a 7065 726d  ding........perm
0170: 6974 2d70 7479 0000 0000 0000 000e 7065  it-pty........pe
0180: 726d 6974 2d75 7365 722d 7263 0000 0000  rmit-user-rc....
0190: 0000 0000 0000 0033 0000 000b 7373 682d  .......3....ssh-
01a0: 6564 3235 3531 3900 0000 20d5 258d f8cb  ed25519... .%...
01b0: 1bda 81d7 9a2f 6b4d 1304 d497 0add 67eb  ...../kM......g.
01c0: 04b9 d5de 47d9 cc0c d5ff 5000 0000 5300  ....G.....P...S.
01d0: 0000 0b73 7368 2d65 6432 3535 3139 0000  ...ssh-ed25519..
01e0: 0040 5864 29cc 18fd 8b1d 19f3 210e c11e  .@Xd).......!...
01f0: 43a9 bde1 90e6 0505 8733 d333 e806 5381  C........3.3..S.
0200: 81b4 d847 6efa 3889 6706 30ea 8117 f2e5  ...Gn.8.g.0.....
0210: 25c4 6e6e 2378 5e27 1c25 7653 3af5 ca9f  %.nn#x^'.%vS:...
0220: 0b05 0a
```

This is an annotated version of the above certificate. Note that all offsets are inclusive, e.g. "0010-0013" refers to a four-byte sequence.

```
# 0000-001b: 0018 6563 6473 ... 7274
string              key type name "ecdsa-sha2-nistp256-cert"
# 001c-003f: 0020 7ee0 cb87 ... aa19
string              nonce
# 0040-004b: 0008 6e69 7374 7032 3536
string              ECDSA curve name "nistp256"
# 004c-0090: 0000 0041 04a5 7c7c ... 5c52
string              ECDSA Q point
```

```
# 0091-0098: ab54 a98c eb1f 0ad2 (12345678901234567890)
uint64              serial number
# 0099-009c: 0000 0001 (SSH2_CERT_TYPE_USER)
uint32              certificate role
# 009d-00b3: 0000 0013 6a6f ... 7267
string              identifier "josef.k@example.org"
# 00b4-00d5: 0000 001e 0000 0007 ... 0790
string              principals
    # 00b8-00c2: 0000 0007 6a6f 7365 662e 6b
    string          principal[0] = "josef.k"
    # 00c3-00d5: 000f 4558 414d 504c 455c 6a6f 7365 662e 6b
    string          principal[1] = "EXAMPLE\josef.k"
# 00d6-00dd: 0000 0000 4d4a 2972
uint64              valid after = 2011/02/03T04:05:06 UTC
# 00de-00e5: 0000 0000 82e9 0790
uint64              valid before = 2039/08/07T06:05:04 UTC
# 00e6-0109: 0020 0000 000d ... 7465
string              critical options
    # 00ea-00fa: 0000 000d 666f 7263 652d 636f 6d6d 616e 64
    string          key[0] = "force-command"
    # 00fb-0109: 0000 000b 0000 0007 6578 6563 7574 65
    string          value[0]
        # 00ff-0109: 0000 0007 6578 6563 7574 65
        string      "execute"
# 010a-0190: 0000 0082 0000 0015 ... 7263
string              extensions
    # 010e-0126: 0000 0015 7065 ... 6e67
    string          key[0] = "permit-X11-forwarding"
    # 0127-012a: 0000 0000
    string          value[0] = (empty)
    # 012b-0145: 0000 0017 7065 726d ... 6e67
    string          key[1] = "permit-agent-forwarding"
    # 0146-0149: 0000 0000
    string          value[1] = (empty)
    # 014a-0163 0000 0016 7065 ... 6e67
    string          key[2] = "permit-port-forwarding"
    # 0164-0167: 0000 0000
    string          value[2] = (empty)
    # 0168-0175: 0000 000a 7065 726d 6974 2d70 7479
    string          key[3] = "permit-pty"
    # 0176-0179: 0000 0000
    string          value[3] = (empty)
    # 017a-018b: 0000 000e 7065 726d 6974 2d75 7365 722d 7263
    string          key[4] = "permit-user-rc"
    # 018c-018f: 0000 0000
    string          value[4] = (empty)
# 0190-0193: 0000 0000
string              reserved = (empty)
```

```
       # 0194-01ca: 0000 0033 0000 000b ... ff50
       string               signature key (ssh-ed25519)
       # 01cb-0222: 0000 0053 0000 000b ... 050a
       string               signature (ssh-ed25519)
```

Author’s Address

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