Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: 31 May 2026

D. Miller
OpenSSH

27 November 2025

SSH Agent Protocol
draft-ietf-sshm-ssh-agent-13

Abstract

This document specifies a key agent protocol for use in the Secure Shell (SSH) protocol.

Note

This note is to be removed before publishing as an RFC.

In the IANA considerations section, please replace "thisRFC" with "RFC" and the assigned number, when known.

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 31 May 2026.

Copyright Notice

Table of Contents

1. Introduction

Secure Shell (SSH) [RFC4251] is a protocol for secure remote connections [RFC4253] and login [RFC4254] over untrusted networks. It supports multiple authentication mechanisms [RFC4252], including public key authentication. This document specifies the protocol for interacting with a key management component, usually referred to as "an agent", that holds private keys. SSH clients (and possibly SSH servers) can invoke the agent via this protocol to perform operations using public and private keys held in the agent.

Holding keys in an agent offers usability and security advantages to loading and unwrapping them at each use, as each key unwrapping may require entry of a pass-phrase. Access to an agent may optionally be forwarded across an SSH connection, thereby allowing remote systems to use stored keys without directly exposing the key material to the remote system. Finally, the agent may be implemented as a dedicated component that presents a smaller attack surface than a key loaded into a full SSH server or client, and which may be subject to special protection from the wider system.

1.1. Background

This section is to be removed before publishing as an RFC.

This agent protocol is already widely used and a de-facto standard, having been implemented by a number of popular SSH clients and servers for many years. The purpose of this document is to describe the protocol as it has been implemented.

1.2. 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.

2. Protocol Overview

The agent protocol is a packetised request-response protocol, solely driven by the client. It consists of a number of requests sent from a client to an agent and a set of reply messages that are sent in response. At no time does the agent send messages except in response to a client request. Replies are sent in order.

These requests include the ability to load keys into an agent, remove some or all keys from an agent and to perform signature operations using previously-loaded keys.

Agents MAY implement support for only a subset of operations or available key types, and MAY additionally refuse arbitrary operations in particular contexts. For example, an agent may allow only clients local to itself to add or remove keys, or make particular subsets of keys available to a given client. For this reason, clients of the agent SHOULD be prepared to fail gracefully if any operation is refused.

2.1. Background

This section is to be removed before publishing as an RFC.

Note that this protocol is separate to and incompatible with the one described in the similarly-named [draft-ietf-secsh-agent-02] which expired in 2004.

2.2. Terminology and units

Henceforth in this document, "agent" will be used to refer to a key management component that implements the responder side of this protocol. "Client" will refer to a tool that implements the requester side of the protocol to communicate with an agent. If it is pertinent that the client in question is a [RFC4251] Secure Shell client, then it will be explicitly referred to as an "SSH client". Similarly "SSH server" will be used to refer to Secure Shell servers.

All encoding data types ("byte", "uint32", "string", etc.) are as specified in Section 5 of [RFC4251]. Additionally, the type "byte[]" without a specified length within the square brackets indicates an unadorned sequence of zero or more bytes where the length is determined by context.

All length units are given in bytes unless otherwise specified.

3. Protocol Messages

Messages consist of a length, type and contents.

```
uint32            length
byte              type
byte[length - 1]  contents
```

In the sections below, the "length" field is omitted. For clarity, the symbolic names of the message types are shown; their numeric values are listed in Section 6.1 below.

3.1. Generic agent responses

The following generic messages may be sent by the agent in response to requests from the client. On success the agent MUST reply either with the single-byte response:

```
byte              SSH_AGENT_SUCCESS
```

or a request-specific success message that may contain additional fields. On failure, the agent MUST reply with the single-byte response:

```
byte              SSH_AGENT_FAILURE
```

or a request-specific failure message that may contain additional fields. SSH_AGENT_FAILURE messages MUST also be sent in reply to requests with unknown or unsupported types.

3.2. Adding keys to the agent

Keys may be added to the agent using the SSH_AGENTC_ADD_IDENTITY or SSH_AGENTC_ADD_ID_CONSTRAINED messages. The latter variant allows adding keys with optional constraints on their usage.

The generic format for the key SSH_AGENTC_ADD_IDENTITY message is:

```
byte            SSH_AGENTC_ADD_IDENTITY
string          key type
byte[]          key data
string          comment
```

Here "key type" is the specified key type name, for example "ssh-rsa" for a RSA key as defined by [RFC4253]. "key data" consists of the public and private components of the key and vary by key type, as specified in sub-sections 3.2.1 through 3.2.4 for commonly used key types. "comment" is a human-readable key name or comment as a UTF-8 string that may serve to identify the key in user-visible messages. This string may be of zero length.

The SSH_AGENTC_ADD_ID_CONSTRAINED is similar, but adds an extra field:

```
byte            SSH_AGENTC_ADD_ID_CONSTRAINED
string          key type
byte[]          key data
string          comment
constraint[]    constraints
```

Constraints are used to place limits on the validity or use of keys. Section 3.2.7 details constraint types and their format. Clients SHOULD prefer the SSH_AGENTC_ADD_IDENTITY message over sending a SSH_AGENTC_ADD_ID_CONSTRAINED with an empty constraints field, though both are valid and equivalent.

An agent MUST reply with SSH_AGENT_SUCCESS if the key was successfully loaded as a result of one of these messages, or SSH_AGENT_FAILURE otherwise.

An agent MAY support only a subset of the key types defined here and MAY support additional key types as described below. If an agent does not recognise the type name in a request to add a key, then it MUST respond with a SSH_AGENT_FAILURE reply.

3.2.1. DSA keys

DSA keys have key type "ssh-dss" and are defined in [RFC4253]. They may be added to the agent using the following message. The "constraints" field is only present for the SSH_AGENTC_ADD_ID_CONSTRAINED message.

```
byte            SSH_AGENTC_ADD_IDENTITY or
                SSH_AGENTC_ADD_ID_CONSTRAINED
string          "ssh-dss"
mpint           p
mpint           q
mpint           g
mpint           y
mpint           x
string          comment
constraint[]    constraints
```

The "p", "q", "g" values are the DSA domain parameters. "y" and "x" are the public and private keys respectively. These values are as defined by Section 4.1 of [FIPS.186-4].

3.2.2. ECDSA keys

ECDSA keys have key types starting with "ecdsa-sha2-" and are defined in [RFC5656]. They may be added to the agent using the following message. The "constraints" field is only present for the SSH_AGENTC_ADD_ID_CONSTRAINED message.

```
byte            SSH_AGENTC_ADD_IDENTITY or
                SSH_AGENTC_ADD_ID_CONSTRAINED
string          key type
string          ecdsa_curve_name
string          Q
mpint           d
string          comment
constraint[]    constraints
```

The values "Q" and "d" are the ECDSA public and private values respectively. Both are defined by Section 6.2 of [FIPS.186-5].

3.2.3. EDDSA keys

[RFC8709] defines Ed25519 and Ed448 with key type names "ssh-ed25519" and "ssh-ed448" respectively. These may be added to the agent using the following message. The "constraints" field is only present for the SSH_AGENTC_ADD_ID_CONSTRAINED message.

```
byte            SSH_AGENTC_ADD_IDENTITY or
                SSH_AGENTC_ADD_ID_CONSTRAINED
string          "ssh-ed25519" or "ssh-ed448"
string          ENC(A)
string          k || ENC(A)
string          comment
constraint[]    constraints
```

The first value is the EDDSA public key ENC(A). The second value is a concatenation of the private key k and the public ENC(A) key (this redundant repetition of the public key is to maintain compatibility with widely deployed implementations). The contents and interpretation of the ENC(A) and k values are defined by Section 3.2 of [RFC8032].

3.2.4. RSA keys

RSA keys have key type "ssh-rsa" and are defined in [RFC4253]. They may be added to the agent using the following message. The "constraints" field is only present for the SSH_AGENTC_ADD_ID_CONSTRAINED message.

```
    byte            SSH_AGENTC_ADD_IDENTITY or
                    SSH_AGENTC_ADD_ID_CONSTRAINED
    string          "ssh-rsa"
    mpint           n
    mpint           e
    mpint           d
    mpint           iqmp
    mpint           p
    mpint           q
    string          comment
    constraint[]    constraints
```

"n" is the public composite modulus. "e" is the public exponent. "d" is the private exponent. "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.186-5].

3.2.5. Other keys

Agents and their clients MAY support additional key types not documented here. Vendor-specific key types MUST use the domain-qualified naming convention defined in Section 6 of [RFC4251] until code-points are allocated by IANA.

3.2.6. Adding keys from a token

Keys hosted on smart-cards or other hardware tokens may be added using the SSH_AGENTC_ADD_SMARTCARD_KEY and SSH_AGENTC_ADD_SMARTCARD_KEY_CONSTRAINED requests. Note that "constraints" field is only included for the SSH_AGENTC_ADD_SMARTCARD_KEY_CONSTRAINED variant of this message.

```
byte            SSH_AGENTC_ADD_SMARTCARD_KEY or
                SSH_AGENTC_ADD_SMARTCARD_KEY_CONSTRAINED
string          token id
string          PIN
constraint[]    constraints
```

Here "token id" is an opaque identifier for the hardware token and "PIN" is an optional password or PIN to unlock the key. The interpretation of "token id" is not defined by the protocol but is left solely up to the agent.

Typically only the public components of any keys supported on a hardware token will be loaded into an agent so, strictly speaking, this message really arranges future private key operations to be delegated to the hardware token in question.

An agent MUST reply with SSH_AGENT_SUCCESS if one or more keys were successfully loaded as a result of one of these messages, or SSH_AGENT_FAILURE if no keys were found. The agent MUST also return SSH_AGENT_FAILURE if the "token id" was not recognised or if the agent doesn’t support token-hosted keys at all.

3.2.7. Key Constraints

A number of constraints may be used in the constrained variants of the key add messages. Each constraint is represented by a type byte followed by zero or more value bytes.

Zero or more constraints may be specified when adding a key with one of the *_CONSTRAINED requests. Multiple constraints are appended consecutively to the end of the request:

```
byte            constraint1_type
byte[]          constraint1_data
byte            constraint2_type
byte[]          constraint2_data
....
byte            constraintN_type
byte[]          constraintN_data
```

To fully parse a constraint, it is necessary to know its structure beforehand and it is not possible to safely recover when an unrecognised constraint is encountered. Given this, if an agent does not recognise or support a requested constraint it MUST abort parsing, refuse the request and return an SSH_AGENT_FAILURE message to the client.

The following constraints are defined.

3.2.7.1. Key lifetime constraint

This constraint requests that the agent limit the key's lifetime by deleting it after the specified duration (in seconds) has elapsed from the time the key was added to the agent.

```
byte            SSH_AGENT_CONSTRAIN_LIFETIME
uint32          seconds
```

3.2.7.2. Key confirmation constraint

This constraint requests that the agent require explicit user confirmation for each private key operation using the key. For example, the agent could present a confirmation dialog before completing a signature operation.

```
byte            SSH_AGENT_CONSTRAIN_CONFIRM
```

3.2.7.3. Constraint extensions

Agents may implement experimental or private-use constraints through a extension constraint that supports named constraints.

```
byte            SSH_AGENT_CONSTRAIN_EXTENSION
string          extension name
byte[]          extension-specific details
```

The extension name MUST consist of a UTF-8 string suffixed by the implementation domain following the naming scheme defined in Section 6 of [RFC4251], e.g. "foo@example.com".

Note, given the above requirement to reject keys with unsupported constraints, a constraint extension is only usable when both client and agent support it. Otherwise, the agent will be required to reject the key. This is desirable, as the constraint extension may specify limits on the key that, if ignored, may result in the key being available in situations the user did not intend (i.e. the agent will fail in a safe way).

3.3. Public key encoding

Keys previously loaded into an agent are referred to by their public key blob, which is 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.

3.4. Removing keys from the agent

A client may request that an agent remove all keys that it stores:

```
byte            SSH_AGENTC_REMOVE_ALL_IDENTITIES
```

On receipt of such a message, an agent MUST delete all keys that it is holding and MUST reply with SSH_AGENT_SUCCESS.

Specific keys may also be removed:

```
byte            SSH_AGENTC_REMOVE_IDENTITY
string          key blob
```

Where "key blob" is the standard public key encoding of the key to be removed (Section 3.3).

An agent MUST reply with SSH_AGENT_SUCCESS if the key was deleted or SSH_AGENT_FAILURE if it was not found.

Token-hosted keys may be removed from an agent using:

```
byte            SSH_AGENTC_REMOVE_SMARTCARD_KEY
string          token id
string          PIN
```

Where "token id" is an opaque identifier for the hardware token and "PIN" is an optional password or PIN (not typically used), both encoded using UTF-8. Requesting deletion of token-hosted keys MUST cause the agent to remove all keys it loaded from the device matching "token id". Note: this operation affects the agent only, it SHOULD NOT caese the keys be deleted from the token itself.

An agent MUST reply with SSH_AGENT_SUCCESS if the key was deleted or SSH_AGENT_FAILURE if it was not found.

3.5. Requesting a list of keys

A client may request a list of keys from an agent using the following message:

```
byte            SSH_AGENTC_REQUEST_IDENTITIES
```

The agent MUST reply with a message with the following preamble.

```
byte            SSH_AGENT_IDENTITIES_ANSWER
uint32          nkeys
```

Where "nkeys" indicates the number of keys to follow. Following the preamble are zero or more keys, each encoded as:

```
string          key blob
string          comment
```

Where "key blob" is the standard public key encoding of the key (Section 3.3) and "comment" is a human-readable comment encoded as a UTF-8 string.

3.6. Private key operations

A client may request the agent perform a private key signature operation using the following message:

```
byte            SSH_AGENTC_SIGN_REQUEST
string          key blob
string          data
uint32          flags
```

Where "key blob" is the key requested to perform the signature (encoded as per Section 3.3), "data" is the data to be signed and "flags" is a bitfield containing the bitwise OR of zero or more signature flags (see below).

If the agent does not support the requested flags, or is otherwise unable or unwilling to generate the signature (e.g. because it doesn't have the specified key, or the user refused confirmation of a constrained key), it MUST reply with an SSH_AGENT_FAILURE message.

On success, the agent MUST reply with:

```
byte            SSH_AGENT_SIGN_RESPONSE
string          signature
```

The signature format is specific to the algorithm of the key type in use. SSH protocol signature formats 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.

3.6.1. Signature flags

Two flags are currently defined for signature request messages: SSH_AGENT_RSA_SHA2_256 and SSH_AGENT_RSA_SHA2_512 (defined in Section 6.3). These two flags are only valid for "ssh-rsa" keys and request that the agent return a signature using the "rsa-sha2-256" or "rsa-sha2-512" signature methods respectively. These signature schemes are defined in [RFC8332].

3.7. Locking and unlocking an agent

The agent protocol supports requesting that an agent temporarily lock itself with a pass-phrase. When locked, an agent MUST suspend processing of sensitive operations (private key signature operations at the very least) until it has been unlocked with the same pass-phrase.

The following message requests agent locking

```
byte            SSH_AGENTC_LOCK
string          passphrase
```

The agent MUST reply with SSH_AGENT_SUCCESS if locked successfully or SSH_AGENT_FAILURE otherwise (e.g. if the agent was already locked).

The following message requests unlocking an agent:

```
byte            SSH_AGENTC_UNLOCK
string          passphrase
```

If the agent is already locked and the pass-phrase matches the one used to lock it then it MUST unlock and reply with SSH_AGENT_SUCCESS. If the agent is already unlocked or if the pass-phrase does not match it MUST reply with SSH_AGENT_FAILURE.

3.8. Extension mechanism

The agent protocol includes an optional extension mechanism that allows vendor-specific and experimental messages to be sent via the agent protocol. Extension requests from the client consist of:

```
byte            SSH_AGENTC_EXTENSION
string          extension type
byte[]          extension request-specific contents
```

The extension type indicates the type of the extension message as a UTF-8 string. Implementation-specific extensions MUST be suffixed by the implementation domain following the extension naming scheme defined in Section 6 of [RFC4251], e.g. "foo@example.com".

An agent that does not support extensions of the supplied type MUST reply with an empty SSH_AGENT_FAILURE message. This reply is also sent by agents that do not support the extension mechanism at all.

The contents of successful extension reply messages are specific to the extension type. Successful extension requests MUST return either SSH_AGENT_SUCCESS on success or an extension-specific response message:

```
byte            SSH_AGENT_EXTENSION_RESPONSE
string          extension type
byte[]          extension response-specific contents
```

Where the extension type is the same as that in the request.

Extension failure SHOULD be signaled using a SSH_AGENT_EXTENSION_FAILURE message:

```
byte            SSH_AGENT_EXTENSION_FAILURE
```

Extensions SHOULD NOT use the standard SSH_AGENT_FAILURE message. This allows failed requests to be distinguished from the extension not being supported.

3.8.1. Query extension

A single, optional extension request "query" is defined to allow a client to query which, if any, extensions are supported by an agent.

```
byte            SSH_AGENTC_EXTENSION
string          "query"
```

If an agent supports the query extension it SHOULD reply with a list of supported extension names.

```
byte            SSH_AGENT_EXTENSION_RESPONSE
string          "query"
string[]        supported extension types
```

4. Connecting to an agent

Agents are exposed to the local system using a connection-oriented endpoint. On Unix-like systems, it is typical to arrange for the agent to listen on a filesystem-based Unix domain socket. On Microsoft Windows, it is usual to use a Windows Named Pipe. Access to these endpoints should be controlled as discussed in Section 8.

In both cases, it is common to expose the name or address of the listening endpoint via an environment variable named "SSH_AUTH_SOCK". Clients of an agent will use this variable to locate and connect to the listening agent. Agents alternately MAY use an implicit mechanism for clients to locate their endpoint, such as a default per-user location.

5. Forwarding access to an agent

The agent protocol may be forwarded over an SSH connection, using the [RFC4254] connection protocol, allowing agent forwarding to be requested for any session channel, using a model that is similar to the connection protocol's support for X11 Forwarding (Section 6.3 of [RFC4254]). This feature is OPTIONAL for SSH protocol and agent implementations.

5.1. Advertising agent forwarding support

Support for agent forwarding may be advertised by an SSH server using the [RFC8308] extension mechanism using the name "agent-forward" in the SSH_MSG_EXT_INFO message.

```
string          "agent-forward"
string          "0" (version)
```

Note that this protocol substantially predates the existence of the [RFC8308] extension mechanism and several widely-deployed SSH implementations that support agent forwarding do not advertise their ability to do so. SSH Clients MAY opportunistically attempt to request agent forwarding in the absence of an [RFC8308] advertisement using the vendor-specific names mentioned below. Likewise, SSH servers MAY implement the vendor-specific names in addition to the one described here.

5.2. Requesting agent forwarding

An SSH client may request agent forwarding for a previously-opened session (Section 6.1 of [RFC4254]) using the following channel request. This request is sent after the channel has been opened, but before a shell, command or subsystem has been executed.

```
byte            SSH_MSG_CHANNEL_REQUEST
uint32          channel_id
string          "agent-req" or "auth-agent-req@openssh.com"
boolean         want_reply
```

Where channel_id is the identifier for an established session channel (as returned from a previous SSH_MSG_CHANNEL_OPEN request, and the want_reply flag indicates whether the SSH server should respond with a confirmation of whether the request was successful (as specified in Section 5.4 of [RFC4254])

If an SSH server accepts this request, typically it will arrange to make a endpoint (e.g. a listening socket) available and advertise this fact to the subordinate session. Most implementations on Unix-like systems do this by providing a user-private listening Unix domain socket and recording its location in an environment variable SSH_AUTH_SOCK.

As mentioned previously, many deployed implementations only support the pre-standardisation "auth-agent-req@openssh.com" request name. The "agent-req" name SHOULD only be used if support was explicitly advertised as per Section 5.1.

5.3. Agent connection requests

After an SSH client has requested that a session have agent forwarding enabled, the SSH server later may request a connection to the forwarded agent. The SSH server does this by requesting a dedicated channel to communicate with the SSH client’s agent.

```
byte            SSH_MSG_CHANNEL_OPEN
string          "agent-connect" or "auth-agent@openssh.com"
uint32          channel_id
uint32          local_window
uint32          local_maxpacket
```

The channel_id, local_window and local_maxpacket fields should be interpreted as specified by Section 5.1 of [RFC4254].

As above, the "agent-connect" open type name SHOULD only be used if support was explicitly advertised as per Section 5.1.

An SSH client SHOULD be prepared to handle multiple concurrent forwarded connections to a client-side agent, otherwise requests to access the agent from the remote side that happen to overlap prior requests may fail. Overlapping requests may occur because the SSH connection protocol [RFC4254] allows multiple user sessions over a single [RFC4253] transport, which may each request use of the agentcw independently and potentially concurrently.

An SSH client MAY accept agent connection requests (subject to authorisation) without a prior agent forwarding request having been made to support the situation where agent forwarding without opening

a session is desired. Similarly, an SSH client MAY continue to accept agent connection requests after the session for which agent forwarding was requested has closed.

An SSH client MUST refuse unauthorised agent connection requests, when agent forwarding is neither requested nor desired by the SSH client but an SSH server sends an agent connection request anyway.

Because the "agent-connect" request contains no identifier to distinguish which session channel originated the connection request, an SSH connection can effectively forward access to only a single SSH client-side agent using this protocol (although there may be multiple concurrent connections to that single agent).

6. Protocol numbers

6.1. Message type numbers

The following numbers are used as message types for requests from the client to the agent.

SSH_AGENTC_REQUEST_IDENTITIES	11
SSH_AGENTC_SIGN_REQUEST	13
SSH_AGENTC_ADD_IDENTITY	17
SSH_AGENTC_REMOVE_IDENTITY	18
SSH_AGENTC_REMOVE_ALL_IDENTITIES	19
SSH_AGENTC_ADD_SMARTCARD_KEY	20
SSH_AGENTC_REMOVE_SMARTCARD_KEY	21
SSH_AGENTC_LOCK	22
SSH_AGENTC_UNLOCK	23
SSH_AGENTC_ADD_ID_CONSTRAINED	25
SSH_AGENTC_ADD_SMARTCARD_KEY_CONSTRAINED	26
SSH_AGENTC_EXTENSION	27

The following numbers are used as message types for replies from the agent to the client.

SSH_AGENT_FAILURE	5
SSH_AGENT_SUCCESS	6
SSH_AGENT_IDENTITIES_ANSWER	12
SSH_AGENT_SIGN_RESPONSE	14
SSH_AGENT_EXTENSION_FAILURE	28
SSH_AGENT_EXTENSION_RESPONSE	29

6.1.1. Reserved message type numbers

The following message type numbers are reserved for implementations that implement support for the legacy SSH protocol version 1: 1-4, 7-10, 15-16 and 24 (inclusive). These message numbers MAY be used by an implementation supporting the legacy protocol but MUST NOT be reused otherwise.

Message number 0 is also reserved and MUST NOT be used.

The range of message numbers 240-255 are reserved for Private Use extensions to the agent protocol and MUST NOT be used by generic implementations.

6.2. Constraint identifiers

The following numbers are used to identify key constraints. These are only used in key constraints and are not sent as message numbers.

SSH_AGENT_CONSTRAIN_LIFETIME	1
SSH_AGENT_CONSTRAIN_CONFIRM	2
SSH_AGENT_CONSTRAIN_EXTENSION	255

The constraint identifier 0 is reserved.

6.3. Signature flags

The following numbers may be present in signature request (SSH_AGENTC_SIGN_REQUEST) messages. These flags form a bit field by taking the logical OR of zero or more flags.

SSH_AGENT_RSA_SHA2_256	0x00000002
SSH_AGENT_RSA_SHA2_512	0x00000004

The flag value 1 is reserved for historical implementations.

7. IANA Considerations

This protocol requires four registries be established, one for message type numbers, one for constraints, one for signature request flags and one for extension request names. Additionally, new codepoints are requested in three existing registries.

7.1. Guidance for Designated Experts

When a Designated Expert (DE) is asked to review additions to the new registries described above (Section 7.2, Section 7.3, Section 7.4 and Section 7.5), they are requested to verify that suitable documentation as described in [RFC5226] exists and is permanently and publicly available. The DE is also requested to check the clarity of purpose and use of the requested code points. The DE should also verify that specifications produced in the IETF that request code points in these registries have been made available to the SSHM working group and the ssh@ietf.org mailing list for review. Requests for code points made for specifications produced outside the IETF should not conflict with active IETF work or prior IETF specifications.

The available number of code points in the SSH agent protocol numbers (Section 7.2) and SSH agent signature flags (Section 7.4) registries are limited, so the DE is requested to ensure the use of code points is very well justified. For the SSH agent protocol numbers, named extension requests (Section 7.5) provide an alternative for most uses with no practical limitation on the number of available code points.

7.2. New registry: SSH agent protocol message type numbers

This registry, titled "SSH agent protocol message type numbers" records the message type numbers for client requests and agent responses. It should be created in the Secure Shell (SSH) Protocol Parameters registry group [IANA-SSH]. Its initial state should consist of the following numbers and reservations. Future message number allocations shall occur via EXPERT REVIEW as per [RFC8126].

Number(s)	Identifier	Reference
0	reserved	thisrfc, Section 6.1.1
1	reserved	thisrfc, Section 6.1.1
2	reserved	thisrfc, Section 6.1.1
3	reserved	thisrfc, Section

		6.1.1
4	reserved	thisrfc, Section 6.1.1
5	SSH_AGENT_FAILURE	thisrfc, Section 6.1
6	SSH_AGENT_SUCCESS	thisrfc, Section 6.1
7	reserved	thisrfc, Section 6.1.1
8	reserved	thisrfc, Section 6.1.1
9	reserved	thisrfc, Section 6.1.1
10	reserved	thisrfc, Section 6.1.1
11	SSH_AGENTC_REQUEST_IDENTITIES	thisrfc, Section 6.1
12	SSH_AGENT_IDENTITIES_ANSWER	thisrfc, Section 6.1
13	SSH_AGENTC_SIGN_REQUEST	thisrfc, Section 6.1
14	SSH_AGENT_SIGN_RESPONSE	thisrfc, Section 6.1
15	reserved	thisrfc, Section

		6.1.1
16	reserved	thisrfc, Section 6.1.1
17	SSH_AGENTC_ADD_IDENTITY	thisrfc, Section 6.1
18	SSH_AGENTC_REMOVE_IDENTITY	thisrfc, Section 6.1
19	SSH_AGENTC_REMOVE_ALL_IDENTITIES	thisrfc, Section 6.1
20	SSH_AGENTC_ADD_SMARTCARD_KEY	thisrfc, Section 6.1
21	SSH_AGENTC_REMOVE_SMARTCARD_KEY	thisrfc, Section 6.1
22	SSH_AGENTC_LOCK	thisrfc, Section 6.1
23	SSH_AGENTC_UNLOCK	thisrfc, Section 6.1
24	reserved	thisrfc, Section 6.1.1
25	SSH_AGENTC_ADD_ID_CONSTRAINED	thisrfc, Section 6.1
26	SSH_AGENTC_ADD_SMARTCARD_KEY_CONSTRAINED	thisrfc, Section 6.1
27	SSH_AGENTC_EXTENSION	thisrfc, Section

		6.1
28	SSH_AGENT_EXTENSION_FAILURE	thisrfc, Section 6.1
29	SSH_AGENT_EXTENSION_RESPONSE	thisrfc, Section 6.1
240-255	Private Use	thisrfc, Section 6.1

Table 1

7.3. New registry: SSH agent key constraint numbers

This registry, titled "SSH agent key constraint numbers" records the message numbers for key use constraints. It should be created in the Secure Shell (SSH) Protocol Parameters registry group [IANA-SSH]. Its initial state should consist of the following numbers. Future key constraint number allocations shall occur via EXPERT REVIEW as per [RFC8126].

Number	Identifier	Reference
1	SSH_AGENT_CONSTRAIN_LIFETIME	thisrfc, Section 6.2
2	SSH_AGENT_CONSTRAIN_CONFIRM	thisrfc, Section 6.2
255	SSH_AGENT_CONSTRAIN_EXTENSION	thisrfc, Section 6.2

Table 2

7.4. New registry: SSH agent signature flags

This registry, titled "SSH agent signature flags" records the values for signature request (SSH_AGENTC_SIGN_REQUEST) flag values. It should be created in the Secure Shell (SSH) Protocol Parameters registry group [IANA-SSH]. Its initial state should consist of the following numbers. Note that as the flags are combined by bitwise OR, all flag values must be powers of two and the maximum available flag value is 0x80000000.

Future signature flag allocations shall occur via EXPERT REVIEW as per [RFC8126].

Number	Identifier	Reference
0x01	reserved	thisrfc, Section 6.3
0x02	SSH_AGENT_RSA_SHA2_256	thisrfc, Section 6.3
0x04	SSH_AGENT_RSA_SHA2_512	thisrfc, Section 6.3

Table 3

7.5. New registry: SSH agent extension request names

This registry, titled "SSH agent extension request names" records the names used in the generic extension request message (SSH_AGENTC_EXTENSION). It should be created in the Secure Shell (SSH) Protocol Parameters registry group [IANA-SSH]. Its initial state should consist of the following names.

Future name allocations shall occur via EXPERT REVIEW as per [RFC8126].

Extension Name	Reference
query	thisrfc, Section 3.8.1

Table 4

7.6. Additions to SSH Extension Names

IANA is requested to insert the following entries into the table Extension Names [IANA-SSH-EXT] in the Secure Shell (SSH) Protocol Parameters registry group [IANA-SSH].

Extension Name	Reference
agent-forward	thisrfc, Section 5.1

Table 5

7.7. Additions to SSH Connection Protocol Channel Request Names

IANA is requested to insert the following entries into the table Connection Protocol Channel Request Names [IANA-SSH-CHANREQ] in the Secure Shell (SSH) Protocol Parameters registry group [IANA-SSH].

Request Type	Reference
agent-req	thisrfc, Section 5.2

Table 6

7.8. Additions to SSH Connection Protocol Channel Types

IANA is requested to insert the following entries into the table Connection Protocol Channel Types [IANA-SSH-CHANTYPE] under Secure Shell (SSH) Protocol Parameters [IANA-SSH].

Request Type	Reference
agent-connect	thisrfc, Section 5.3

Table 7

8. Security Considerations

The agent is a service that is tasked with retaining and providing controlled access to what are typically long-lived login authentication credentials. It is by nature a sensitive and trusted software component. Moreover, the agent protocol itself does not include any authentication or transport security; ability to communicate with an agent is usually sufficient to invoke it to perform private key operations.

Since being able to access an agent is usually sufficient to perform private key operations, it is critically important that the agent only be exposed to its owner and their authorised delegates. On Unix-like systems this may be achieved via filesystem permissions on the agent socket and/or identity checks on the client connected to a socket (e.g. SO_PEERCRED on some Unix-like systems). On Windows, access to a named pipe may controlled by attaching a security descriptor at the time of its creation.

The primary design intention of an agent is that an attacker with unprivileged access to their victim's agent should be prevented from gaining a copy of any keys that have been loaded into it. This may not preclude the attacker from stealing use of those keys (e.g. if they have been loaded without a confirmation constraint).

Given this, the agent should, as far as possible, prevent its memory being read by other processes to prevent theft of loaded keys. This typically includes disabling debugging interfaces and preventing process memory dumps on abnormal termination.

Another, more subtle, means by which keys may be stolen are via cryptographic side-channels. Private key operations may leak information about the contents of keys via differences in timing, power use or by side-effects in the memory subsystems (e.g. CPU caches) of the host running the agent. For the case of a local attacker and an agent holding unconstrained keys, the only limit on the number of private key operations the attacker may be able to observe is the rate at which the CPU can perform signatures. This grants the attacker an almost ideal oracle for side-channel attacks. While a full treatment of side-channel attacks is beyond the scope of this specification, agents SHOULD use cryptographic implementations that are resistant to side-channel attacks and MAY take additional measures to hide the actual time spent processing private key operations. Failure to do so may expose keys to recovery through these side-channels.

Forwarding access to a local agent over an SSH connection (Section 5) inherently creates a transitive trust relationship. SSH implementations SHOULD NOT forward use of an agent by default, as doing so could expose access to the user's keys to untrusted hosts they connect to. Agents SHOULD implement additional controls over key visibility and use for forwarded agent connections, otherwise the user has only an all-or-nothing choice over whether to forward an agent.

Implementation of token/smartcard-hosted keys requires some care too. On some systems, tokens may invoked by providing a path to shared library that must be loaded to make use of keys hosted on the device (a path to a library for a particular PKCS#11 module, for example). Loading a shared library on most platforms implies automatic execution of code in that library in the address space of the process that loads it. To avoid loading of potentially-hostile code, agents that support loading token-hosted keys via library path SHOULD ensure that only trusted token provider libraries are loadable. Additionaly agents SHOULD ensure that loaded token library code cannot gain access to other keys loaded in the agent and MAY disallow remote clients from loading token keys entirely. Protection for existing

keys from tokey library code may be achieved by loading the token library into a separate process to the agent and arranging for the agent to invoke token operations to this process via IPC.

Finally, with respect to the agent locking functionality in Section 3.7, an agent SHOULD take countermeasures against brute-force guessing attacks against the pass-phrase. This may take the form of enforced delays when an unlock attempt is made with an incorrect password (potentially increasing for subsequent failures), a lockout period where the agent refuses to accept further requests after some threshold of failed unlock attempts has been made and/or deletion of all keys held by the agent after a threshold of failed unlock attempts.

9. Implementation Status

This section is to be removed before publishing as an RFC.

Note to editor: please also remove the orphaned reference to RFC7942.

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 projects maintain an implementation of this protocol:

OpenSSH OpenSSH is the originating implementation of this protocol and has supported it since 2000.

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

PuTTY PuTTY is a popular SSH client implementation for multiple

platforms that has included a compatible agent client since 2001.

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

Dropbear Dropbear is an SSH client and server implementation for Unix- like systems. It has supported the agent protocol since 2005.

Website: https://matt.ucc.asn.au/dropbear/dropbear.html

Paramiko Paramiko is an SSH client and server implementation in the Python programming language. It has supported an agent protocol implementation since 2005.

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

Golang x/crypto/ssh/agent The Go programming language project has supported an implementation of this protocol in its external "x" repository since 2015.

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

This list is not exhaustive.

10. References

10.1. 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>.

[RFC4251] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Protocol Architecture", RFC 4251, DOI 10.17487/RFC4251, January 2006, <https://www.rfc-editor.org/info/rfc4251>.

[RFC4253] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, DOI 10.17487/RFC4253, January 2006, <https://www.rfc-editor.org/info/rfc4253>.

[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>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 5226, DOI 10.17487/RFC5226, May 2008, <https://www.rfc-editor.org/info/rfc5226>.

[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>.

[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>.

[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>.

[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>.

[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>.

10.2. Informative References

[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>.

[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>.

[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>.

[IANA-SSH-CHANREQ] IANA, "Connection Protocol Channel Types", <https://www.iana.org/assignments/ssh-parameters/>.

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

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

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

[draft-ietf-secsh-agent-02] Ylonen, T., Rinne, T. J., and S. Lehtinen, "Secure Shell Authentication Agent Protocol", January 2004, <https://datatracker.ietf.org/doc/html/draft-ietf-secsh-agent-02>.

Acknowledgments

This protocol was designed and first implemented by Markus Friedl, based on a similar protocol for an agent to support the legacy SSH version 1 by Tatu Ylonen.

Thanks to Simon Tatham, Niels M旦ller, James Spencer, Simon Josefsson, Matt Johnston, Jakub Jelen, Rich Salz, Caspar Schutijser, Florian Obser, Martin Thomson, Deb Cooley and Tero Kivinen who reviewed and helped improve this document.

Author’s Address

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