Workgroup: Internet Engineering Task Force
Internet-Draft: draft-miller-ssh-agent-14
Published: 15 April 2024
Intended Status: Informational
Expires: 17 October 2024
Authors: D. Miller
OpenSSH

SSH Agent Protocol

Abstract

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

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 17 October 2024.

Copyright Notice

Table of Contents

1. Introduction

Secure Shell (SSH) is a protocol for secure remote connections and login over untrusted networks. It supports multiple authentication mechanisms, including public key authentication. This document describes the protocol for interacting with an agent that holds private keys. Clients (and possibly servers) can use 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. Moreover, the agent implements a simple protocol and presents a smaller attack surface than a key loaded into a full SSH server or client.

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.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Protocol Overview

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

Agents MAY implement support for only a subset of operations or available key types, and MAY additionally refuse abitrary operations in particular contexts. For example, an agent may allow only local clients of an agent 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.

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

3. Protocol Messages

All values in the agent protocol are encoded using the SSH wire representations specified by [RFC4251]. Messages consist of a length, type and contents.

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

3.1. Generic server responses

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

```
byte              SSH_AGENT_SUCCESS
```

or a request-specific success message. On failure, the agent may reply with:

```
byte              SSH_AGENT_FAILURE
```

SSH_AGENT_FAILURE messages are also sent in reply to requests with unknown 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 contents
string            key comment
```

Here "type" is the specified key type name, for example "ssh-rsa" for a RSA key as defined by [RFC4253]. "contents" consists of the public and private components of the key and vary by key type, they are listed below for standard and commonly used key types. "comment" is an optional human-readable key name or comment as a UTF-8 string that may serve to identify the key in user-visible messages.

The SSH_AGENTC_ADD_ID_CONSTRAINED is similar, but adds a extra field:

```
byte            SSH_AGENTC_ADD_ID_CONSTRAINED
string          type
byte[]          contents
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.

An agent should 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 fail gracefully and 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 [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 [FIPS.186-4].

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 "key 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. The contents and interpretation of the ENC(A) and k values are defined by [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 "key 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. "p" and "q" are its constituent private prime factors. "e" is the public exponent. "iqmp" is the inverse of "q" modulo "p". All these values except "iqmp" (which can be calculated from the others) are defined by [FIPS.186-4].

3.2.5. Other keys

Agents and their 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].

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          id
string          PIN
constraint[]    constraints
```

Here "id" is an opaque identifier for the hardware token and "PIN" is an optional password on PIN to unlock the key. The interpretation of "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 should 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 should 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 and 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
```

If an agent does not recognise or support a requested constraint it MUST refuse the request and return a 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 4.2 of [RFC4251], e.g. "foo@example.com".

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 shall delete all keys that it is holding and 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 shall reply with SSH_AGENT_SUCCESS if the key was deleted or SSH_AGENT_FAILURE if it was not found.

Smartcard keys may be removed using:

```
byte            SSH_AGENTC_REMOVE_SMARTCARD_KEY
string          reader id
string          PIN
```

Where "reader id" is an opaque identifier for the smartcard reader and "PIN" is an optional password or PIN (not typically used). Requesting deletion of smartcard-hosted keys SHOULD cause the agent to remove all keys loaded from that smartcard.

An agent shall 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 shall 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 a SSH_AGENT_FAILURE message.

On success, the agent shall 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. 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 should 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 shall 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 should unlock and reply with SSH_AGENT_SUCCESS. If the agent is unlocked or if the the pass-phrase does not match it should reply with SSH_AGENT_FAILURE. An agent SHOULD take countermeasures against brute-force guessing attacks against the pass-phrase.

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 should be suffixed by the implementation domain following the extension naming scheme defined in Section 4.2 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. Extension requests may return SSH_AGENT_SUCCESS on success or the 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 the SSH_AGENT_EXTENSION_FAILURE code - 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 is should reply with a list of supported extension names.

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

4. Forwarding access to an agent

The agent protocol may be forwarded over a 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.

Note that the deployed integration with the SSH protocol uses vendor-specific names.

4.1. Advertising agent forwarding support

Support for agent forwarding may be advertised by a SSH protocol 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. Clients MAY opportunitically attempt to request agent forwarding in the absence of an [RFC8308] advertisment using the vendor-specific names mentioned below. Likewise, servers MAY implement the vendor-specific names in addition to the one described here.

4.2. Requesting agent forwarding

A 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
string          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 server should respond with a confirmation of whether the request was successful (as specified in Section 5.4 of [RFC4254])

If a 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 former, pre-standardisation "auth-agent-req@openssh.com" request name. The latter "agent-req" name SHOULD only be used if support was explicitly advertised as per Section 4.1.

4.3. Agent connection requests

After a client has requested that a session have agent forwarding enabled, the server later may request a connection to the forwarded agent. The server does this by requesting a dedicated channel to communicate with the 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 latter "agent-connect" open type name SHOULD only be used if support was explicitly advertised as per Section 4.1.

A client SHOULD be prepared to handle multiple concurrent active agent connections. A 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, a client MAY continue to accept agent connection requests after the session for which agent forwarding was requested has closed.

A client MUST refuse unauthorised agent connection requests, i.e. when agent forwarding was not desired or requested but a a server sends an agent connection request anyway.

Note the connection request provides no way to identify which session channel a given agent connection request relates to. This implies that a SSH connection can functionally forward access to only a single client-side agent concurrently using this protocol.

5. Protocol numbers

5.1. Message numbers

The following numbers are used 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 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

5.1.1. Reserved message numbers

The following message numbers are reserved for implementations that implement support for the legacy SSH protocol version 1: 1-4, 7-9 and 24 (inclusive). These message numbers MAY be used by an implementation supporting the legacy protocol but MUST NOT be reused otherwise.

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

5.2. 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

5.3. 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	2
SSH_AGENT_RSA_SHA2_512	4

The flag value 1 is reserved for historical implementations.

6. IANA Considerations

This protocol requires three registries be established, one for message numbers, one for constraints and one for signature request flags.

6.1. New registry: SSH agent protocol numbers

This registry, titled "SSH agent protocol numbers" records the message numbers for client requests and agent responses. Its initial state should consist of the following numbers and reservations. Future message number allocations shall require specification in the form of an RFC (RFC REQUIRED as per [RFC5226]).

Number(s)	Identifier	Reference
1	reserved	Section 5.1.1
2	reserved	Section 5.1.1
3	reserved	Section 5.1.1
4	reserved	Section 5.1.1
5	SSH_AGENT_FAILURE	Section 5.1
6	SSH_AGENT_SUCCESS	Section 5.1
7	reserved	Section 5.1.1
8	reserved	Section 5.1.1
9	reserved	Section 5.1.1
10	reserved	Section 5.1.1
11	SSH_AGENTC_REQUEST_IDENTITIES	Section 5.1
12	SSH_AGENT_IDENTITIES_ANSWER	Section 5.1
13	SSH_AGENTC_SIGN_REQUEST	Section 5.1
14	SSH_AGENT_SIGN_RESPONSE	Section 5.1
15	reserved	Section 5.1.1
16	reserved	Section 5.1.1
17	SSH_AGENTC_ADD_IDENTITY	Section 5.1
18	SSH_AGENTC_REMOVE_IDENTITY	Section 5.1
19	SSH_AGENTC_REMOVE_ALL_IDENTITIES	Section 5.1
20	SSH_AGENTC_ADD_SMARTCARD_KEY	Section 5.1
21	SSH_AGENTC_REMOVE_SMARTCARD_KEY	Section 5.1
22	SSH_AGENTC_LOCK	Section 5.1
23	SSH_AGENTC_UNLOCK	Section 5.1
24	reserved	Section 5.1.1
25	SSH_AGENTC_ADD_ID_CONSTRAINED	Section 5.1
26	SSH_AGENTC_ADD_SMARTCARD_KEY_CONSTRAINED	Section 5.1
27	SSH_AGENTC_EXTENSION	Section 5.1
28	SSH_AGENT_EXTENSION_FAILURE	Section 5.1
29	SSH_AGENT_EXTENSION_RESPONSE	Section 5.1
240-255	Reserved for organizational use	Section 5.1

Table 1

6.2. New registry: SSH agent key constraint numbers

This registry, titled "SSH agent key constraint numbers" records the message numbers for key use constraints. Its initial state should consist of the following numbers. Future constraint number

allocations shall require specification in the form of an RFC (RFC REQUIRED as per [RFC5226]).

Number	Identifier	Reference
1	SSH_AGENT_CONSTRAIN_LIFETIME	Section 5.2
2	SSH_AGENT_CONSTRAIN_CONFIRM	Section 5.2
255	SSH_AGENT_CONSTRAIN_EXTENSION	Section 5.2

Table 2

6.3. 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. 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 constraint number allocations shall require specification in the form of an RFC (RFC REQUIRED as per [RFC5226]).

Number	Identifier	Reference
0x01	reserved	Section 5.3
0x02	SSH_AGENT_RSA_SHA2_256	Section 5.3
0x04	SSH_AGENT_RSA_SHA2_512	Section 5.3

Table 3

6.4. 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). Its initial state should consist of the following names.

Future constraint number allocations shall require specification in the form of an RFC (RFC REQUIRED as per [RFC5226]).

Extension Name	Reference
query	Section 3.8.1

Table 4

6.5. Additions to SSH Extension Names

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

Extension Name	Reference
agent-forward	Section 4.1

Table 5

6.6. 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] under Secure Shell (SSH) Protocol Parameters [RFC4250].

Extension Name	Reference
agent-forward	Section 4.2

Table 6

6.7. 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 [RFC4250].

Extension Name	Reference
agent-connect	Section 4.3

Table 7

7. 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.

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 direct theft of loaded keys. This typically include 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.

Forwarding access to a local agent over a SSH connection (Section 4) inherently creates a transitive trust relationship. SSH implementations SHOULD NOT forward use of an agent by default and MAY implement additional controls over key visibility and use for forwarded agent connections.

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>.

[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, July 2013.

8.2. Informative References

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

[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>.

Acknowledgements

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 Möller and James Spencer who reviewed and helped improve this document.

Author's Address

Damien Miller
OpenSSH

Email: djm@openssh.com
URI: https://www.openssh.com/