Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: April 3, 2018

D. Miller
OpenSSH

September 30, 2017

SSH Agent Protocol
draft-miller-ssh-agent-01

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 April 3, 2018.

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.

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. 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.

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

4. Protocol Messages

4.1. 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.

4.2. 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 4.2.6 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.

4.2.1. 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].

4.2.2. 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].

4.2.3. ED25519 keys

Ed25519 keys have key type "ssh-ed25519" and are defined in [I-D.ietf-curdle-ssh-ed25519]. 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-ed25519"
string                  ENC(A)
string                  k || ENC(A)
string                  comment
constraint[]            constraints
```

The first value is the 32 byte Ed25519 public key "ENC(A)". The second value is a concatenation of the 32 byte private key "k" and 32 byte public "ENC(A)" key. The contents and interpretation of the "ENC(A)" and "k" values are defined by [I-D.irtf-cfrg-eddsa].

4.2.4. 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].

4.2.5. 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.

4.2.6. 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.

4.2.6.1. 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
```

4.2.6.2. 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
```

4.2.6.3. 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".

4.3. 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. SSH protocol key encodings are defined in [RFC4253] for "ssh-rsa" and "ssh-dss" keys, in [RFC5656] for "ecdsa-sha2-*" keys and in [I-D.ietf-curdle-ssh-ed25519] for "ssh-ed25519" keys.

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 will 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.

4.4. 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 wire encoding of the public key and "comment" is a human-readable comment encoded as a UTF-8 string.

4.5. 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, "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 [I-D.ietf-curdle-ssh-ed25519] for "ssh-ed25519" keys.

4.5.1. 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 [I-D.ietf-curdle-rsa-sha2].

4.6. 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 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.

4.7. 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 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 some other extension-specific message.

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.

4.7.1. 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.

If an agent supports the "query" extension is should reply with a list of supported extension names.

```
byte                    SSH_AGENT_SUCCESS
string[]                extension type
```

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_ID_CONSTRAINED	25
SSH_AGENTC_ADD_SMARTCARD_KEY	20
SSH_AGENTC_REMOVE_SMARTCARD_KEY	21
SSH_AGENTC_LOCK	22
SSH_AGENTC_UNLOCK	23
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_EXTENSION_FAILURE	28
SSH_AGENT_IDENTITIES_ANSWER	12
SSH_AGENT_SIGN_RESPONSE	14

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.

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	3

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. 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 who reviewed and helped improve this document.

7. IANA Considerations

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

7.1. 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	Identifier	Reference
1	reserved	Section 5.1
2	reserved	Section 5.1
3	reserved	Section 5.1
4	reserved	Section 5.1
5	SSH_AGENT_FAILURE	Section 5.1
6	SSH_AGENT_SUCCESS	Section 5.1
7	reserved	Section 5.1
8	reserved	Section 5.1
9	reserved	Section 5.1
10	reserved	Section 5.1
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
16	reserved	Section 5.1
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
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

Initial registry state: SSH agent protocol numbers

7.2. 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
3	SSH_AGENT_CONSTRAIN_EXTENSION	Section 5.2

Initial registry state: SSH agent key constraint numbers

7.3. 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

Initial registry state: SSH agent signature flags

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.

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 in to 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.

9. Normative References

[FIPS.186-4]
National Institute of Standards and Technology, "Digital Signature Standard (DSS)", FIPS PUB 186-4, July 2013.

[I-D.ietf-curdle-rsa-sha2]
bider, d., "Use of RSA Keys with SHA-2 256 and 512 in Secure Shell (SSH)", draft-ietf-curdle-rsa-sha2-10 (work in progress), August 2017.

[I-D.ietf-curdle-ssh-ed25519]
Harris, B. and L. Velvindron, "Ed25519 public key algorithm for the Secure Shell (SSH) protocol", draft-ietf-curdle-ssh-ed25519-01 (work in progress), August 2017.

[I-D.irtf-cfrg-eddsa]
Josefsson, S. and I. Liusvaara, "Edwards-curve Digital Signature Algorithm (EdDSA)", draft-irtf-cfrg-eddsa-08 (work in progress), August 2016.

[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>.

[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>.

Author's Address

Damien Miller
OpenSSH

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