Network Working Group T. Ylonen
Internet-Draft T. Kivinen
Expires: May 10, 2002 SSH Communications Security Corp
M. Saarinen
University of Jyvaskyla
T. Rinne
S. Lehtinen
SSH Communications Security Corp
November 9, 2001

SSH Protocol Architecture
draft-ietf-secsh-architecture-10.txt

Abstract

SSH is a protocol for secure remote login and other secure network services over an insecure network. This document describes the architecture of the SSH protocol, as well as the notation and terminology used in SSH protocol documents. It also discusses the SSH algorithm naming system that allows local extensions. The SSH protocol consists of three major components: The Transport Layer

Protocol provides server authentication, confidentiality, and integrity with perfect forward secrecy. The User Authentication Protocol authenticates the client to the server. The Connection Protocol multiplexes the encrypted tunnel into several logical channels. Details of these protocols are described in separate documents.

Table of Contents

1. Introduction 3
2. Specification of Requirements 3
3. Architecture 3
3.1 Host Keys 3
3.2 Extensibility 5
3.3 Policy Issues 5
3.4 Security Properties 6
3.5 Packet Size and Overhead 6
3.6 Localization and Character Set Support 7
4. Data Type Representations Used in the SSH Protocols 8
5. Algorithm Naming 10
6. Message Numbers 10
7. IANA Considerations 11
8. Security Considerations 12
9. Trademark Issues 12
10. Additional Information 12
References 12
Authors' Addresses 13
Full Copyright Statement 15

1. Introduction

SSH is a protocol for secure remote login and other secure network services over an insecure network. It consists of three major components:

- o The Transport Layer Protocol [SSH-TRANS] provides server authentication, confidentiality, and integrity. It may optionally also provide compression. The transport layer will typically be run over a TCP/IP connection, but might also be used on top of any other reliable data stream.
- o The User Authentication Protocol [SSH-USERAUTH] authenticates the client-side user to the server. It runs over the transport layer protocol.
- o The Connection Protocol [SSH-CONNECT] multiplexes the encrypted tunnel into several logical channels. It runs over the user authentication protocol.

The client sends a service request once a secure transport layer connection has been established. A second service request is sent after user authentication is complete. This allows new protocols to be defined and coexist with the protocols listed above.

The connection protocol provides channels that can be used for a wide range of purposes. Standard methods are provided for setting up secure interactive shell sessions and for forwarding ("tunneling") arbitrary TCP/IP ports and X11 connections.

2. Specification of Requirements

All documents related to the SSH protocols shall use the keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" to describe requirements. They are to be interpreted as described in [RFC-2119].

3. Architecture

3.1 Host Keys

Each server host SHOULD have a host key. Hosts MAY have multiple host keys using multiple different algorithms. Multiple hosts MAY share the same host key. If a host has keys at all, it MUST have at least one key using each REQUIRED public key algorithm (currently DSS [FIPS-186]).

The server host key is used during key exchange to verify that the client is really talking to the correct server. For this to be possible, the client must have a priori knowledge of the server's public host key.

Two different trust models can be used:

- o The client has a local database that associates each host name (as typed by the user) with the corresponding public host key. This method requires no centrally administered infrastructure, and no third-party coordination. The downside is that the database of name-to-key associations may become burdensome to maintain.
- o The host name-to-key association is certified by some trusted certification authority. The client only knows the CA root key, and can verify the validity of all host keys certified by accepted CAs.

  The second alternative eases the maintenance problem, since ideally only a single CA key needs to be securely stored on the client. On the other hand, each host key must be appropriately certified by a central authority before authorization is possible. Also, a lot of trust is placed on the central infrastructure.

The protocol provides the option that the server name - host key association is not checked when connecting to the host for the first time. This allows communication without prior communication of host keys or certification. The connection still provides protection against passive listening; however, it becomes vulnerable to active man-in-the-middle attacks. Implementations SHOULD NOT normally allow such connections by default, as they pose a potential security problem. However, as there is no widely deployed key infrastructure available on the Internet yet, this option makes the protocol much more usable during the transition time until such an infrastructure emerges, while still providing a much higher level of security than that offered by older solutions (e.g. telnet [RFC-854] and rlogin [RFC-1282]).

Implementations SHOULD try to make the best effort to check host keys. An example of a possible strategy is to only accept a host key without checking the first time a host is connected, save the key in a local database, and compare against that key on all future connections to that host.

Implementations MAY provide additional methods for verifying the correctness of host keys, e.g. a hexadecimal fingerprint derived from the SHA-1 hash of the public key. Such fingerprints can easily be verified by using telephone or other external communication channels.

All implementations SHOULD provide an option to not accept host keys that cannot be verified.

We believe that ease of use is critical to end-user acceptance of security solutions, and no improvement in security is gained if the

new solutions are not used. Thus, providing the option not to check the server host key is believed to improve the overall security of the Internet, even though it reduces the security of the protocol in configurations where it is allowed.

3.2 Extensibility

We believe that the protocol will evolve over time, and some organizations will want to use their own encryption, authentication and/or key exchange methods. Central registration of all extensions is cumbersome, especially for experimental or classified features. On the other hand, having no central registration leads to conflicts in method identifiers, making interoperability difficult.

We have chosen to identify algorithms, methods, formats, and extension protocols with textual names that are of a specific format. DNS names are used to create local namespaces where experimental or classified extensions can be defined without fear of conflicts with other implementations.

One design goal has been to keep the base protocol as simple as possible, and to require as few algorithms as possible. However, all implementations MUST support a minimal set of algorithms to ensure interoperability (this does not imply that the local policy on all hosts would necessary allow these algorithms). The mandatory algorithms are specified in the relevant protocol documents.

Additional algorithms, methods, formats, and extension protocols can be defined in separate drafts. See Section Algorithm Naming (Section 5) for more information.

3.3 Policy Issues

The protocol allows full negotiation of encryption, integrity, key exchange, compression, and public key algorithms and formats. Encryption, integrity, public key, and compression algorithms can be different for each direction.

The following policy issues SHOULD be addressed in the configuration mechanisms of each implementation:

- o Encryption, integrity, and compression algorithms, separately for each direction. The policy MUST specify which is the preferred algorithm (e.g. the first algorithm listed in each category).
- o Public key algorithms and key exchange method to be used for host authentication. The existence of trusted host keys for different public key algorithms also affects this choice.
- o The authentication methods that are to be required by the server for each user. The server's policy MAY require multiple

authentication for some or all users. The required algorithms MAY depend on the location where the user is trying to log in from.

- o The operations that the user is allowed to perform using the connection protocol. Some issues are related to security; for example, the policy SHOULD NOT allow the server to start sessions or run commands on the client machine, and MUST NOT allow connections to the authentication agent unless forwarding such connections has been requested. Other issues, such as which TCP/IP ports can be forwarded and by whom, are clearly issues of local policy. Many of these issues may involve traversing or bypassing firewalls, and are interrelated with the local security policy.

3.4 Security Properties

The primary goal of the SSH protocol is improved security on the Internet. It attempts to do this in a way that is easy to deploy, even at the cost of absolute security.

- o All encryption, integrity, and public key algorithms used are well-known, well-established algorithms.
- o All algorithms are used with cryptographically sound key sizes that are believed to provide protection against even the strongest cryptanalytic attacks for decades.
- o All algorithms are negotiated, and in case some algorithm is broken, it is easy to switch to some other algorithm without modifying the base protocol.

Specific concessions were made to make wide-spread fast deployment easier. The particular case where this comes up is verifying that the server host key really belongs to the desired host; the protocol allows the verification to be left out (but this is NOT RECOMMENDED). This is believed to significantly improve usability in the short term, until widespread Internet public key infrastructures emerge.

3.5 Packet Size and Overhead

Some readers will worry about the increase in packet size due to new headers, padding, and MAC. The minimum packet size is in the order of 28 bytes (depending on negotiated algorithms). The increase is negligible for large packets, but very significant for one-byte packets (telnet-type sessions). There are, however, several factors that make this a non-issue in almost all cases:

- o The minimum size of a TCP/IP header is 32 bytes. Thus, the increase is actually from 33 to 51 bytes (roughly).
- o The minimum size of the data field of an Ethernet packet is 46 bytes [RFC-894]. Thus, the increase is no more than 5 bytes. When Ethernet headers are considered, the increase is less than 10 percent.

o The total fraction of telnet-type data in the Internet is negligible, even with increased packet sizes.

The only environment where the packet size increase is likely to have a significant effect is PPP [RFC-1134] over slow modem lines (PPP compresses the TCP/IP headers, emphasizing the increase in packet size). However, with modern modems, the time needed to transfer is in the order of 2 milliseconds, which is a lot faster than people can type.

There are also issues related to the maximum packet size. To minimize delays in screen updates, one does not want excessively large packets for interactive sessions. The maximum packet size is negotiated separately for each channel.

3.6 Localization and Character Set Support

For the most part, the SSH protocols do not directly pass text that would be displayed to the user. However, there are some places where such data might be passed. When applicable, the character set for the data MUST be explicitly specified. In most places, ISO 10646 with UTF-8 encoding is used [RFC-2279]. When applicable, a field is also provided for a language tag [RFC-1766].

One big issue is the character set of the interactive session. There is no clear solution, as different applications may display data in different formats. Different types of terminal emulation may also be employed in the client, and the character set to be used is effectively determined by the terminal emulation. Thus, no place is provided for directly specifying the character set or encoding for terminal session data. However, the terminal emulation type (e.g. "vt100") is transmitted to the remote site, and it implicitly specifies the character set and encoding. Applications typically use the terminal type to determine what character set they use, or the character set is determined using some external means. The terminal emulation may also allow configuring the default character set. In any case, the character set for the terminal session is considered primarily a client local issue.

Internal names used to identify algorithms or protocols are normally never displayed to users, and must be in US-ASCII.

The client and server user names are inherently constrained by what the server is prepared to accept. They might, however, occasionally be displayed in logs, reports, etc. They MUST be encoded using ISO 10646 UTF-8, but other encodings may be required in some cases. It is up to the server to decide how to map user names to accepted user names. Straight bit-wise binary comparison is RECOMMENDED.

For localization purposes, the protocol attempts to minimize the number of textual messages transmitted. When present, such messages typically relate to errors, debugging information, or some externally configured data. For data that is normally displayed, it SHOULD be possible to fetch a localized message instead of the transmitted message by using a numerical code. The remaining messages SHOULD be configurable.

4. Data Type Representations Used in the SSH Protocols

byte

A byte represents an arbitrary 8-bit value (octet) [RFC-1700]. Fixed length data is sometimes represented as an array of bytes, written byte[n], where n is the number of bytes in the array.

boolean

A boolean value is stored as a single byte. The value 0 represents FALSE, and the value 1 represents TRUE. All non-zero values MUST be interpreted as TRUE; however, applications MUST NOT store values other than 0 and 1.

uint32

Represents a 32-bit unsigned integer. Stored as four bytes in the order of decreasing significance (network byte order). For example, the value 699921578 (0x29b7f4aa) is stored as 29 b7 f4 aa.

uint64

Represents a 64-bit unsigned integer. Stored as eight bytes in the order of decreasing significance (network byte order).

string

Arbitrary length binary string. Strings are allowed to contain arbitrary binary data, including null characters and 8-bit characters. They are stored as a uint32 containing its length (number of bytes that follow) and zero (= empty string) or more bytes that are the value of the string. Terminating null characters are not used.

Strings are also used to store text. In that case, US-ASCII is used for internal names, and ISO-10646 UTF-8 for text that might be displayed to the user. The terminating null character SHOULD NOT normally be stored in the string.

For example, the US-ASCII string "testing" is represented as 00 00 00 07 t e s t i n g. The UTF8 mapping does not alter the encoding of US-ASCII characters.

mpint

Represents multiple precision integers in two's complement format, stored as a string, 8 bits per byte, MSB first. Negative numbers have the value 1 as the most significant bit of the first byte of the data partition. If the most significant bit would be set for a positive number, the number MUST be preceded by a zero byte. Unnecessary leading bytes with the value 0 or 255 MUST NOT be included. The value zero MUST be stored as a string with zero bytes of data.

By convention, a number that is used in modular computations in Z_n SHOULD be represented in the range 0 &lt= x &lt n.

Examples:

value (hex)	representation (hex)
0	00 00 00 00
9a378f9b2e332a7	00 00 00 08 09 a3 78 f9 b2 e3 32 a7
80	00 00 00 02 00 80
-1234	00 00 00 02 ed cc
-deadbeef	00 00 00 05 ff 21 52 41 11

name-list

A string containing a comma separated list of names. A name list is represented as a uint32 containing its length (number of bytes that follow) followed by a comma-separated list of zero or more names. A name MUST be non-zero length, and it MUST NOT contain a comma (','). Context may impose additional restrictions on the names; for example, the names in a list may have to be valid algorithm identifier (see Algorithm Naming below), or [RFC-1766] language tags. The order of the names in a list may or may not be significant, also depending on the context where the list is is used. Terminating NUL characters are not used, neither for the individual names, nor for the list as a whole.

Examples:

value	representation (hex)
(), the empty list	00 00 00 00
("zlib")	00 00 00 04 7a 6c 69 62

```
      ("zlib", "none")   00 00 00 09 7a 6c 69 62 2c 6e 6f 6e 65
```

5. Algorithm Naming

The SSH protocols refer to particular hash, encryption, integrity, compression, and key exchange algorithms or protocols by names. There are some standard algorithms that all implementations MUST support. There are also algorithms that are defined in the protocol specification but are OPTIONAL. Furthermore, it is expected that some organizations will want to use their own algorithms.

In this protocol, all algorithm identifiers MUST be printable US-ASCII non-empty strings no longer than 64 characters. Names MUST be case-sensitive.

There are two formats for algorithm names:

- o Names that do not contain an at-sign (@) are reserved to be assigned by IETF consensus (RFCs). Examples include `3des-cbc', `sha-1', `hmac-sha1', and `zlib' (the quotes are not part of the name). Names of this format MUST NOT be used without first registering them. Registered names MUST NOT contain an at-sign (@) or a comma (,).
- o Anyone can define additional algorithms by using names in the format name@domainname, e.g. "ourcipher-cbc@ssh.com". The format of the part preceding the at sign is not specified; it MUST consist of US-ASCII characters except at-sign and comma. The part following the at-sign MUST be a valid fully qualified internet domain name [RFC-1034] controlled by the person or organization defining the name. It is up to each domain how it manages its local namespace.

6. Message Numbers

SSH packets have message numbers in the range 1 to 255. These numbers have been allocated as follows:

```
  Transport layer protocol:

    1 to 19    Transport layer generic (e.g. disconnect, ignore, debug,
               etc.)
    20 to 29   Algorithm negotiation
    30 to 49   Key exchange method specific (numbers can be reused for
               different authentication methods)
```

User authentication protocol:

```
  50 to 59   User authentication generic
  60 to 79   User authentication method specific (numbers can be
             reused for different authentication methods)
```

Connection protocol:

```
  80 to 89   Connection protocol generic
  90 to 127  Channel related messages
```

Reserved for client protocols:

```
  128 to 191 Reserved
```

Local extensions:

```
  192 to 255 Local extensions
```

7. IANA Considerations

Allocation of the following types of names in the SSH protocols is assigned by IETF consensus:

- o encryption algorithm names,
- o MAC algorithm names,
- o public key algorithm names (public key algorithm also implies encoding and signature/encryption capability),
- o key exchange method names, and
- o protocol (service) names.

These names MUST be printable US-ASCII strings, and MUST NOT contain the characters at-sign ('@'), comma (','), or whitespace or control characters (ASCII codes 32 or less). Names are case-sensitive, and MUST NOT be longer than 64 characters.

Names with the at-sign ('@') in them are allocated by the owner of DNS name after the at-sign (hierarchical allocation in [RFC-2343]), otherwise the same restrictions as above.

Each category of names listed above has a separate namespace. However, using the same name in multiple categories SHOULD be avoided to minimize confusion.

Message numbers (see Section Message Numbers (Section 6)) in the range of 0..191 should be allocated via IETF consensus; message numbers in the 192..255 range (the "Local extensions" set) are

reserved for private use.

8. Security Considerations

Special care should be taken to ensure that all of the random numbers are of good quality. The random numbers SHOULD be produced with safe mechanisms discussed in [RFC-1750].

When displaying text, such as error or debug messages to the user, the client software SHOULD replace any control characters (except tab, carriage return and newline) with safe sequences to avoid attacks by sending terminal control characters.

Not using MAC or encryption SHOULD be avoided. The user authentication protocol is subject to man-in-the-middle attacks if the encryption is disabled. The SSH protocol does not protect against message alteration if no MAC is used.

9. Trademark Issues

As of this writing, SSH Communications Security Oy claims ssh as its trademark. As with all IPR claims the IETF takes no position regarding the validity or scope of this trademark claim.

10. Additional Information

The current document editor is: Darren.Moffat@Sun.COM. Comments on this internet draft should be sent to the IETF SECSH working group, details at: http://ietf.org/html.charters/secsh-charter.html

References

[FIPS-186]   Federal Information Processing Standards Publication, ., "FIPS PUB 186, Digital Signature Standard", May 1994.

[RFC0854]   Postel, J. and J. Reynolds, "Telnet Protocol Specification", STD 8, RFC 854, May 1983.

[RFC0894]   Hornig, C., "Standard for the transmission of IP datagrams over Ethernet networks", STD 41, RFC 894, Apr 1984.

[RFC1034]   Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, Nov 1987.

[RFC1134]   Perkins, D., "Point-to-Point Protocol: A proposal for multi-protocol transmission of datagrams over Point-

to-Point links", RFC 1134, Nov 1989.

[RFC1282] Kantor, B., "BSD Rlogin", RFC 1282, December 1991.

[RFC1700] Reynolds, J. and J. Postel, "Assigned Numbers", STD 2, RFC 1700, October 1994.

[RFC1750] Eastlake, D., Crocker, S. and J. Schiller, "Randomness Recommendations for Security", RFC 1750, December 1994.

[RFC1766] Alvestrand, H., "Tags for the Identification of Languages", RFC 1766, March 1995.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2279] Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 2279, January 1998.

[RFC2434] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 2434, October 1998.

[SSH-ARCH] Ylonen, T., "SSH Protocol Architecture", I-D draft-ietf-architecture-09.txt, July 2001.

[SSH-TRANS] Ylonen, T., "SSH Transport Layer Protocol", I-D draft-ietf-transport-11.txt, July 2001.

[SSH-USERAUTH] Ylonen, T., "SSH Authentication Protocol", I-D draft-ietf-userauth-11.txt, July 2001.

[SSH-CONNECT] Ylonen, T., "SSH Connection Protocol", I-D draft-ietf-connect-11.txt, July 2001.

Authors' Addresses

Tatu Ylonen
SSH Communications Security Corp
Fredrikinkatu 42
HELSINKI FIN-00100
Finland

EMail: ylo@ssh.com

Tero Kivinen
SSH Communications Security Corp
Fredrikinkatu 42
HELSINKI FIN-00100
Finland

EMail: kivinen@ssh.com

Markku-Juhani O. Saarinen
University of Jyvaskyla

Timo J. Rinne
SSH Communications Security Corp
Fredrikinkatu 42
HELSINKI FIN-00100
Finland

EMail: tri@ssh.com

Sami Lehtinen
SSH Communications Security Corp
Fredrikinkatu 42
HELSINKI FIN-00100
Finland

EMail: sjl@ssh.com

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.