SSH - Secure Login Connections over the Internet

Tatu Ylonen
SSH Communications Security Ltd, Finland

SSH – Secure Login Connections over the Internet

Tatu Ylönen <ylo@ssh.fi>
SSH Communications Security Ltd.
Tekniikantie 12, FIN-02150 ESPOO, Finland
Tel. (intl) +358-0-4354 3205 fax +358-0-4354 3206

June 7, 1996

Abstract

SSH provides secure login, file transfer, X11, and TCP/IP connections over an untrusted network. It uses cryptographic authentication, automatic session encryption, and integrity protection for transferred data. RSA is used for key exchange and authentication, and symmetric algorithms (e.g., IDEA or three-key triple-DES) for encrypting transferred data.

SSH is intended as a replacement for the existing `rsh`, `rlogin`, `rcp`, `rdist`, and `telnet` protocols. SSH is currently (March 1996) being used at thousands of sites in at least 50 countries. Its users include top universities, research laboratories, many major corporations, and numerous smaller companies and individuals.

The SSH protocol can also be used as a generic transport layer encryption mechanism, providing both host authentication and user authentication, together with privacy and integrity protection.

1 Introduction

The Internet has become the most economical means for communication between two remote sites. Its uses include communicating with clients, connecting remote offices, file transfer, remote systems administration, banking services, working at home, and many others.

However, the Internet does not provide any protection for the transmitted information, and can become an information security nightmare for companies connected to it. Firewalls and access controls such as one-time passwords do not fully solve the problem, as it is easy to record and analyze any transmitted data or to hijack an already established connection and use it to attack machines inside the firewall.

The threats from the Internet include:

- Network monitoring: it is easy to record passwords, financial data, private messages, or corporate secrets from the network.
- Connection hijacking: it is possible to hijack a connection without either party noticing it, insert new commands at the command prompt, and remove the output of those commands from the output sent to the user. The same mechanisms can, for example, be used to manipulate remote banking connections to make wire transfers with a different sum and account than what the user thinks (and sees). Having the accounts protected by one-time passwords does not help.
- Routing spoofing: standard routing protocols and commonly used router configurations permit anyone in the world to reconfigure routings. This can be used to bring connections into networks through which they do not normally go, and where they can be hijacked.
- DNS (domain name server spoofing): active network-level attacks can be used to make name servers return whatever data is beneficial for further attacks. Verification by reverse-mapping does not help. The same holds for practically all widely used network services.
- Denial of service attacks: here the purpose is to prevent others from using a particular service. The simplest implementation of this attack is to overload the target machine with requests; however, more subtle forms are available, such as reconfiguring the routers so that packets no longer go to the machine, hijacking connections to the machine and returning erroneous results, etc.

The current IP protocol does not in any way guarantee any aspects of information security (e.g., authentication, privacy, data integrity). As higher level protocols are mostly based on the assumption that the lower level protocol can be trusted – and as this is not the case – the higher level protocols aren't any more reliable. If security is needed, it must be entirely implemented on the application level.

An acceptable solution must guarantee at the same time authentication of both ends of the connection, secrecy of transmitted information, and integrity of transmitted data. For example, if only authentication and integrity (but no secrecy) is provided, the user is likely to eventually type a password to another machine or service, which will then be shown in the clear.

Strong encryption seems to be the only solution to network security. Since several major governments have demonstrated growing interest in economic espionage (including, e.g., United States, Russia, France, and Japan), commercial systems are now faced with some of the most skilled and resourceful opponents in the world, and must be designed using the strongest possible methods to be of any use. Increasing economic significance will also lure interest from criminal organizations, which are certainly well enough funded to break e.g. DES-level encryption methods without much trouble [12].

2 An Overview of SSH Secure Remote Login

SSH permits secure login connections and file transfer over the Internet or other untrusted networks. Cryptographic algorithms are used to authenticate both ends of the connection, to automatically encrypt all transmitted data, and to protect the integrity of data. Values returned by services such as DNS or network protocols (e.g., TCP/IP [10]) are considered only advisory, and are validated using cryptography. SSH also automatically and securely forwards X11 connections from the remote machine, and can be configured to forward arbitrary TCP/IP ports. It can also be used for secure file transfer.

3 The SSH Protocol

SSH uses a packet-based binary protocol that works on top of any transport that will pass a stream of binary data. Normally, TCP/IP is used as the transport, but the current implementation also permits using an arbitrary proxy program to pass data to/from the server, and includes direct support for SOCKS and FWTK based firewalls.

The packet mechanism and related authentication, key exchange, encryption, and integrity mechanisms implement a transport layer security mechanism, which is then used to implement the remote login functionality. An attacker is limited to breaking the connection.

Every transmitted packet starts with random padding, followed by (optionally compressed) packet type, packet data, and integrity protection data.

The entire packet is encrypted using a suitable algorithm, such as IDEA-CFB [2, 9], 3DES-CBC [9], or an RC4[1] [9] equivalent algorithm. The packet type and data fields can be compressed with the gzip algorithm before encryption. Compression reduces the amount of transmitted data to about a third for typical interactive sessions.

Integrity protection is currently (March 1996) provided by including CRC32 [1] of the packet under encryption. However, it is being replaced by HMAC-SHA; see Section 5. If tampering is detected, the error is logged, the user is notified, and the connection is terminated.

On the transport, each encrypted packet is prefixed by the length of the packet data, excluding padding (the total length on the wire is the given length rounded up to a multiple of eight bytes in such a way that the length of padding is 1-8 bytes).

The SSH protocol works on top of the packet-level protocol, and proceeds in the following phases:

1. The client opens a connection to the server. (Note that an attacker may cause the connection to actually go to a different machine.)

2. The server sends its public RSA host key and another public RSA key ("server key") that changes every hour. The client compares the received host key against its own database of known host keys (in future, it will validate the host key using a public key infrastructure; however, at present no such infrastructure exists).

   At present, SSH is not able to validate keys for hosts that it does not already know. It will normally accept the key of an unknown host and store it in its database for future reference (this makes SSH usable in practice in most environments). However, SSH can also be configured to refuse access to any hosts whose key is not known.

3. The client generates a 256 bit random number using a cryptographically strong random number generator, and chooses an encryption algorithm from those supported by the server (normally IDEA or three-key 3DES). The client encrypts

[1] RC4 is a trademark of RSA Data Security, Inc.

the random number (session key) with RSA using both the host key and the server key, and sends the encrypted key to the server.

The purpose of the host key is to bind the connection to the desired server host (only the server can decrypt the encrypted session key). The server key is used to make decrypting recorded historic traffic impossible after the server key has been changed (usually every hour) in the event that the host key becomes compromised. The host key is normally a 1024 bit RSA key, and the server key is 768 bits. Both keys are generated using a cryptographically strong random number generator.

4. The server decrypts the RSA encryptions and recovers the session key. Both parties start using the session key (until this point, all traffic has been unencrypted on the packet level). The server sends an encrypted confirmation to the client. Receipt of the confirmation tells the client that the server was able to decrypt the key, and thus holds the proper private keys.

   At this point, the server machine has been authenticated, and transport-level encryption and integrity protection are in use.

5. The user is authenticated to the server. This can happen in a number of ways; the dialog is driven by the client which sends requests to the server. The first request always declares the user name to log in as. The server responds to each request with either "success" (no further authentication is needed) or "failure" (further authentication is required).

   Currently supported authentication methods are:

   - Traditional password authentication. The password is transmitted over the encrypted channel, and thus cannot be seen by outsiders.
   - A combination of traditional .rhosts or hosts.equiv authentication and RSA-based host authentication. Host authentication works by the server generating a 256 bit challenge, encrypting it with the client's public host key, and sending the encrypted challenge to the client. The client decrypts the challenge, and computes MD5 [7] of the challenge and other information that binds the returned value to the particular session. The client then sends this value to the server; the server makes the corresponding computations and compares the values.
   - Pure RSA authentication. The idea is that possession of a particular private RSA key serves as authentication. The server has a list of accepted public keys. The client requests authentication by a particular key, and the server responds with a challenge similar to that in RhostsRSA authentication.
   - Support is also included e.g. for Security Dynamics SecurID cards. Adding new authentication methods is easy.

6. After authentication has been successful, a preparatory phase begins. In this phase, the client sends requests that prepare for the actual session. Such requests include allocation of a pseudo-tty, X11 forwarding, TCP/IP forwarding, etc. Adding new preparatory operations is easy.

   After all other requests, the client sends a request to start the shell or to execute a given command. This message causes both sides to enter the interactive session.

7. During the interactive session, both sides are allowed to send packets asynchronously. The packets may contain data, open requests for X11 connections, forwarded TCP/IP ports, or the agent, etc. Finally at some point the client usually sends an EOF message. When the user's shell or command exits, the server sends its exit status to the client, and the client acknowledges the message and closes the connection.

More information about the protocol can be found in [13].

3.1 X11 and TCP/IP Forwarding

SSH can automatically forward the connection to the user's X server over the secure channel. Forwarding works by creating a proxy X server at the remote machine by allocating the next available TCP/IP port number above 6001 (these correspond to X display numbers so that the port corresponding to display n is $6000 + n$). The SSH server then listens for connections on this port, forwards the connection request and any data over the secure channel, and makes a connection to the real X server from the SSH client. The DISPLAY variable is automatically set to point to the proper value. Note that forwardings can be chained, permitting safe use of X applications over an arbitrary chain of SSH connections.

SSH also automatically stores Xauthority data [8] on the server. In fact, the client generates a random MIT-MAGIC-COOKIE-1 authentication cookie, and sends this cookie to the server, which stores it in `.Xauthority`. When a connection is made, the client verifies that the authority data matches the generated random data, and replaces it with the real data. The motivation for sending a fake cookie is that old cookies left at the server are useless after logout (many users keep the same terminal open for months at a time, and may briefly log into dozens of machines during that time; it is important to not leave the cookies lying around in all of these machines).

TCP/IP forwarding works similarly: the server listens for a socket on the desired port, forwards the request and data over the secure channel, and makes the connection to the specified target port from the other side. There is no authentication for forwarded TCP/IP connections.

3.2 The Authentication Agent

SSH supports using an authentication agent. The agent is a program that runs in the user's local machine (or, in future, on a smartcard connected to it). The agent holds the user's private RSA keys. It never gives out the private keys, but accepts authentication requests and gives back suitable answers.

In the Unix environment, the agent communicates with SSH using an open file handle that is inherited by all children of the agent process (the agent is started as a parent of the user's shell). Other users cannot get access to the agent, and even for root it is fairly difficult to send requests to a file descriptor held by some process. Different mechanisms are used on other operating systems.

SSH can forward the connection to the agent to another process running on the server machine (such as another SSH connection). In this way, it is possible to go through an arbitrarily long chain of machines, located anywhere around the world, without the authentication keys ever leaving the agent.

4 Cryptographic Methods Used in SSH

SSH attempts to provide strong security without making normal use any more difficult than necessary. Its security relies on cryptographic methods.

SSH uses RSA [6, 9] for host authentication and user authentication. Host keys and user authentication keys are normally 1024 bits.

The server key that changes every hour is 768 bits by default. It is used to protect intercepted historical sessions from being decrypted if the host key is later compromised. The server key is never saved on disk.

Key exchange is performed by encrypting the 256-bit session key twice using RSA. It is padded with non-zero random bytes before each encryption (according to PKCS#1 [5]). Server host authentication happens implicitly with the key exchange (the idea is that only the holder of the valid private key can decrypt the session key, and receipt of the encrypted confirmation tells the client that the session key was successfully decrypted).

Client host authentication and RSA user authentication are done using a challenge-response exchange, where the response is MD5 of the decrypted challenge plus data that binds the result to a specific session (host key and anti-spoofing cookie).

The key exchange transfers 256 bits of keying data to the server. Different encryption methods use varying amounts of the key: IDEA-CFB uses 128 bits, 3DES-CBC 168 bits, RC4-equivalent 128 bits per direction, and DES-CBC 56 bits. The reasons for using IDEA in CFB mode is mainly historical; the new protocol (Section 5) will use IDEA-CBC instead.

Transmitted data is currently protected against modification by computing a CRC32 of all packet data (including random padding) before encryption. The checksum and all packet data are encrypted. Presumably it will be difficult for an attacker to modify the plaintext data so that the checksum still matches without breaking the encryption first. (The integrity mechanism has changed in the new protocol; see Section 5.)

All random numbers used in SSH are generated with a cryptographically strong generator. SSH has a pool of 8192 bits of randomness. The first time it is started, it uses several commands to gather entropy from the system (on Unix, "ps laxww", "ps -al", "ls -alni /tmp/.", "w", "netstat -s", "netstat -an", and "netstat -in"). The entropy is mixed into the pool, stirring the pool frequently. The stirring involves encrypting the pool twice using MD5 in CBC mode so that every bit of the pool depends on every other bit. Additional noise is obtained from various system parameters (e.g., disk I/O counts, page swapping counts, interrupt counts, CPU usage) every time the pool is stirred, and if `/dev/random` is available, 128 bits of noise are taken from there every few minutes and stirred into the pool.

5 The New Protocol

The SSH protocol is currently undergoing major changes. The protocol will be split to two levels, a generic secure transport layer mechanism and a high-level SSH protocol.

5.1 The New Transport Layer Protocol

The new transport layer protocol has been designed to be flexible, allowing negotiation of all algorithms and parameters, simple, secure, easily verifiable, and fast. It performs a full algorithm negotiation, key exchange, and mutual host authentication in a total of 1.5 round-trip times typical, and 2.5 round-trip times worst case. A minimal number of round-trips will become increasingly important in future as network bandwidth increases but the speed of light remains constant. Mobile computing, in particular, will put strong demands on the number of roundtrips; over a GSM phone, for example, a round-trip is around a second.

There have been several cryptographic improvements. All data exchanged during key exchange is authenticated. HMAC-MD5 or HMAC-SHA outside encryption are used for data integrity protection. IDEA is now used in CBC mode. All data, including the packet length, is now encrypted (except the MAC). Keys are re-exchanged periodically. The protocol can also interface with a public key infrastructure.

5.2 The New SSH Protocol

The new SSH protocol runs over the transport layer protocol, which provides a secure channel. The SSH protocol performs user authentication, session management, and handshaking for multiple simultaneous connections (forwarded X11 connections, etc).

User authentication now permits the client to send authentication requests without waiting for responses from the server after each request. This reduces round-trips. Additionally, whenever an authentication request fails (or is insufficient), the server will tell the client which authentication methods can continue the dialog. This permits the server to guide the client through a multi-phase authentication according to the server's per-user policy. The server can require multiple authentications.

All authentication methods that require user input have been merged under one interactive authentication type. This handles passwords, one-time passwords, SecurID cards, and other such methods. The user basically converses with the server using a simple text-based protocol. The protocol does, however, permit dialog-based windowed implementation and local editing at the client.

The new protocol also supports proper flow control for individual channels (e.g., forwarded X11 clients). This will prevent a runaway program from jamming the entire connection. Details of the new protocol are still being specified as of this writing.

6 The Current Implementation

SSH was first published on the Internet in July 1995. Since then, it has been ported to a number of platforms and there have been several other improvements.

SSH currently runs on almost all Unix variants, including e.g. AIX, BSD, Convex, DGUX, HPUX, IRIX, Linux, Mach3, OSF/1, SysV, Solaris, SunOS, Ultrix, and Unicos. A commercial Windows version is available from Data Fellows, and a Macintosh version is due in the fall 1996. A free OS/2 version is also available.

The current Unix version supports SOCKS and FWTK based firewalls, and permits using an arbitrary proxy program to make the connection. In most environments, it can be installed simply by

```
./configure
make
make install
```

7 Performance

Performance of SSH can divided into two important parameters: startup time and transfer rate.

The startup time means the time from starting the SSH client to the moment when first data bytes are transferred. The startup time is on the order of a second on 486 or Pentium class machines connected to an ethernet, and on the order of a few seconds for long-distance connections.

Transfer rate means the number of bytes per second that can be transmitted over the secure channel. In the case of SSH, it depends on the encryption algorithm used. On 486-class machines, the rate is 1-2 megabits/second for IDEA, 3-4 megabits/second for DES, and about 5 megabits per second for RC4-equivalent. The rate is almost directly proportional to the speed of the machines; some faster machines run RC4-equivalent in software at speeds exceeding 40 megabits per second.

To summarize, the encryption speed on even slower modern machines is sufficient to fill an ethernet network. Most of the time, transfer rate is not limited by encryption but by the transfer rate of the network. Furthermore, on long-distance connections SSH can be substantially faster than `telnet` or `rlogin`, due to compression of transferred data.

8 Conclusion

SSH solves one of the most acute security problems on the Internet: that of securely logging from one machine to another, and securely transferring files between machines. It does this in a way that is convenient and completely transparent to users. At the same time, it automates passing the X11 connection, and makes using X11 over long distance connections secure.

SSH uses strong cryptography to achieve this goal. Its fundamental principle is that the network or any of its services cannot be trusted. Usability in normal environments has been a major design concern from the beginning, and SSH attempts to make things as easy for normal users as possible while still maintaining a sufficient level of security. For the most security conscious environments, SSH can be configured to never trust the network, and fail if it cannot e.g. verify the host key of the remote host.

Experience has shown that the CPU overhead caused by strong encryption is negligible. One need not try to justify why to encrypt; doing so costs almost nothing. However, not using strong encryption in all communications can have severe consequences. Also, the strongest available encryption methods should be used, as they are no more expensive than weak methods. Weak encryption will just make transmitted data available to foreign intelligence agencies and criminal organizations.

SSH is currently (June 1996) being used at thousands or tens of thousands of sites in at least 50 countries around the world. There are about one thousand addresses on the mailing list, and many of those are redistribution aliases or newsgroup gateways. The SSH WWW pages are accessed about 1000-2000 times every day (about once every minute). During about a period of about ten days (examined in February 1996), accesses came from about 6000 hosts (many of them WWW proxy/cache servers) in 55 top-level domains. The actual number of people using SSH is not known.

SSH is freely available for non-commercial use. Its WWW home page, including pointers to ftp sites and commercial versions, is available at `http://www.cs.hut.fi/ssh`.

References

[1] J. Campbell. *C Programmer's Guide to Serial Communications*, Sams, 1993.

[2] Lai, X. *On the Design and Security of Block Ciphers.* ETH Series in Information Processing, vol. 1, Hartung-Gorre Verlag, Konstantz, 1992.

[3] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, and L. Jones. *SOCKS Protocol Version 5*, RFC 1928, 1996.

[4] Mockapetris, P. *Domain Names – Concepts and Facilities*, RFC 1034, Internet Engineering Task Force, 1987.

[5] *Public Key Cryptography Standards, #1.* RSA Laboratories. Available for anonymous ftp at ftp.rsa.com.

[6] Rivest, R., Shamir, A., and Adleman, L. M. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, vol. 21, no. 2, 1978, pp. 120-126.

[7] Rivest, R. *The MD5 Message Digest Algorithm*, RFC 1321, Internet Engineering Task Force, 1992.

[8] Scheifler, R. *X Window System Protocol.* X Consortium Standard, Version 11, Release 6. Laboratory of Computer Science, Massachusetts Institute of Technology, 1994.

[9] Schneier, Bruce. *Applied Cryptography*, 2nd edition. John Wiley & Sons, 1996.

[10] Stevens, W. Richard. *TCP/IP Illustrated. Volume 1: The Protocols.* Addison-Wesley, 1994.

[11] TIS Firewall Toolkit, Trusted Information Systems Inc., 1993.

[12] Wiener, M. J. *Efficient DES Key Search.* Technical Report TR-244, School of Computer Science, Carleton University, 1994.

[13] Ylönen, Tatu. *The SSH (Secure Shell) Remote Login Protocol*, 1996. Available on the Internet from the SSH Home Page at http://www.cs.hut.fi/ssh. Also included in the SSH distribution.