# ssh-protocol-tools

## Contents



# Introduction

What is it ? SSH is the Secure Shell, a popular powerful software based approach to network security. Whenever data is sent by a computer to the network, SSH automatically encrypts it. When the data reaches its intended recipient, SSH automatically decrypts it. The result is transparent encryption: users can work normally unaware that their communications are safely encrypted on the network. In addition, SSH uses modern secure encryption algorithms and is effective enough to be found within mission critical application at major corporations.

SSH has a client/server architecture. An SSH server program typically installed and run by a system administrator, accepts or rejects incoming connections to its host computer. Users then run SSH client programs, typically on other computers to make requests of the SSH server, such as - logging in, sending files, or executing commands. All communication between the clients and the servers are security encrypted and protected from modification.

Our description is simplified but should give you a general idea of what SSH is and does. An SSH based product might include clients, servers or both. Unix products generally container both clients and servers, those on other platforms are usually just clients, though Windows based servers do exist.

If you are a Unix user, think of SSH as a secure form of the Unix r-commands, `rsb` (remote shell), `rlogin` (remote login) and `rcp` (remote copy). In fact the original SSH for Unix includes the similarly named commands `ssb`, `scp` and `slogin`, as secure, drop in replacements for the r-commands

## What SSH is NOT

Although SSH stands for Secure Shell, it is not a true shell in the sense of the Unix `Bourne shell` (bash) and `C shell` (sh). It is not a command interpreter nor does it provide wildcard expansion command history and so forth. Rather, SSH creates a channel for running a shell on a remote computer in the manner of the Unix `rsb` command. With end-to-end encryption between the local and remote computer. SSH is also not a complete security solution - but then again nothing is. It will not protect computers from active break-in attempts or denial of service attacks, and it wont eliminate other hazards such as viruses. It does however provide a robust and user friendly encryption and authentication.

## The SSH protocol

SSH is a protocol first and foremost, not a product, It is a specification of how to conduct secure communication over a network. The SSH protocol covers authentication, encryption and the integrity of data transmitted over a network.

- Authentication - reliably determines someone's identity. If you try to log into an account on a remote computer, SSH asks for digital proof of your identify. If you pass the test, you may log in; otherwise SSH rejects the connection.
- Encryption - scrambles data so it is unintelligible expect to the intended recipient. This protects your data as it passes over the network.
- Integrity - guarantees the data traverling over the network arrives unaltered. If a third party captures and modifies you data in transit SSH detects this fact.

In short, SSH, makes network connections between computers with strong guarantees that the parties on both ends of the connection are genuine. It also ensures that any data passing over these connections arrives unmodified and unread by eavesdroppers

## Protocols, Products, Clients and Confusion

SSH based products are those that implement the SSH protocol, for many flavors of UNIX, Windows, Macintosh and other operating systems. Bot freely distributable and commercial products are available. The first SSH product, was created for Unix, was simply called SSH. This causes confusion because SSH is also the name of the protocol. Some people called it Unix SSH, but other unix based implementations are not available so the name is not satisfactory.

## Overview of SSH features

As already mentioned SSH mostly revolves around secure login, data transmission and integrity

### Secure login

Let us suppose that we have accounts on several computers on the Internet. Typically you connect from a home PC to your ISP, and then use a telnet program to log into your accounts on other computers. Unfortunately telnet transmits your username and password in plaintext, over the internet, where a malicious third party can intercept them. Additionally your entire telnet session is readable by a network snooper. SSH completely solves this problem. Rather than running the insecure telnet program and protocol, you run the SSH program ssh, to login to the remote account or computer. This is done with the following `ssh -l smith host.example.com -p 22`

The clients authenticates you to the remote computer's SSH server using an encrypted connection meaning that your username and password are encrypted before they leave the local machine. The SSH server then logs you in and your entire login session is encrypted as it travels between the client and server. Because the encryption is transparent you will not notice any differences between telnet and the SSH client.

## Secure data transfer

Suppose you have two computers or machines on the internet and wish to transfer files between them, The file contains trade secrets about your business, however that must be kept from prying eyes. A traditional file-transfer program such as ftp, rcp or even email, does not provide a secure solution. A third party can intercept and read packets as it as they travel over the network. To get around this problem you can encrypt the file on the first machine, with a program, then send it over via traditional means, and decrypt the file on the second machine. Using SSH, the file can be transferred, securely between the machines with a single secure copy command - `scp myfile user@host.com`. When transmitted by scp the file is automatically encrypted as it leaves the first machine, and decrypted as it arrives on the second machine

## Secure remote execution

Suppose you are a system admin who needs to run the same command on many computers. You would like to view the active processes for each user on four different computers. On a local area network using the Unix command /usr/ucb/w. Traditionally one could use `rsb`, assuming that the `rsb` daemon, `rsbd` is configured properly on the remote computers. `rsh $machine /usr/usb/w`. This will work, however it is insecure, The results of executing the `/usr/usb/` binary are transmitted over the network as plaintext. If you consider this information sensitive, the risk might be unacceptable, Worse the `rsb` authentication mechanism is very insecure, and easily subverted. Instead using the `ssb` command one can do - `ssb $machine /usr/ucb/w`. Where the `$machine` is a `sh` variable which represents the host name of the machine to connect to, that could be a FQDN, or an IP address.

The syntax for both commands is nearly identical and the visible output is identical, but under the hood, the command and its results are encrypted, sent, and then decrypted, and strong authentication techniques will be used to ensure secure user login.

## Keys and agents

Suppose you have accounts on many computers on a network. For security reasons you prefer different passwords on all accounts, but remembering so many passwords is difficult. It is also a security problem in itself. The more often you type a password, the more likely you mistakenly type it in the wrong place. SSH has various authentication mechanisms and the most secure is based on keys rather than passwords. Keys

are a small blob of bits that uniquely identifies an SSH user. For security a key is kept encrypted, and it may be used only after entering a secret pass phrase to decrypt it.

Using keys together with a program called an authentication agent, SSH can authenticate you to all computers accounts securely without requiring you to memorize many different passwords, or enter them repeatedly. The way it works is like that:

- In advance (only once) you place files called public key files into the remote computers. These enable you to access the remote machines. These public keys identify `you as you` on the remote machines, you would later try to access
- On your local machine, invoke the ssh-agent program which runs in the background. This daemon will hold the keys `in memory`, and only needs to be started once, when you login to your local machine for the day, this can be automated on most Unix systems, using `services`, or other tools such as `systemd`.
- Choose the key (or keys) you will need during your login session. These `have to correspond to the keys added in step one` to the remote machines, they will be used to authenticate you when a connection is made.
- Load the keys into the agent, with the `ssh-add program`. This requires knowledge or each key's secret passphrase in order to load it into the ssh-agent daemon, this is however done only once and until the agent daemon works, the keys are loaded

At this point you have an ssh-agent program running on your local machine, holding your secret keys, in memory. You are now done. You have password less access to all your remote accounts that contain your public key files. Say goodbye to all the tedium of retyping passwords. The setup lasts until you log out from the local machine or terminate the ssh-agent

## Access control

Suppose you want to permit another person to use your computer account but only for certain purposes, for example while you are out of town you would like your secretary to read your email but not to do anything else in your accounts. With SSH you can give your secretary access to your account without revealing or changing your password, and with only the ability to run the email program. No system administrator privileges are required so setup this restricted access.

## Port forwarding

SSH can increase the security of other TCP based apps such as telnet ftp and the X Window subsystem. A technique called port forwarding or tunneling re routes, TCP connection to pass through an SSH connection transparently encrypting it end to end, Port forwarding can also pass such apps through network firewalls that otherwise prevent their use. Suppose you are logged into a remote machine away from work and want to access the internal news server at your office. The network is connected to the internet but a network firewall blocks incoming connections to the most ports particularly port 119 the news post. The firewall does allow incoming SSH connections however, since the SSH protocol is secure enough that even that network, with its rabidly paranoid system administrators trust it. SSH can establish a secure tunnel on an arbitrary local TCP port say port 3002 to the news port on the remote host, the command might look a bit cryptic at this early stage but here it is - `ssh -L 302:localhost:119 news.yoyodyne.com` This says `ssh` please establish a secure connection from TCP port 3002 on my local machine to TCP port 119 the news port on the server - news.yoyodyne.com, So in order to securely configure your news reading program to connect to port 3002 on your local machine. The secure tunnel created by ssh automatically communicates with the news server on that host, and the news traffic passing through the tunnel is protected by encryption.

## History of SSH

SSH1 and the SSH-1 protocol were developed in 1995, by a researched at the Helsinki University of Technology in Finland. After his university network was the victim of password sniffing atatck earlier that year, he whipped up SSH1 for himself. When beta versions started gaining attention however he realized that his security product could be put to wider use. In July 1995 SSH1 was released to the public as free software with source code, permitting people to copy and use the program without cost, by the end of hte year and estimated 20k users in 50 countries had adopted SSH1 and he was fending off 150 email messages per day requesting support. In response he founded the SSH communications security, Ltd (SCS).

Also in 1995 Yolonen documented the SSH1 protocol as an internet engineering task force (IETF) internet draft which essentially described the operation of the SSH1 software after the fact. It was somewhat ad-hoc protocol with a number of problems and limitations discovered as the software grew in popularity. These problems could not be fixed without losing backward compatibility so in 1996 SCS introduced a new major version of the protocol SSH 2.0 or SSH2 that incorporates new algorithms and is incompatible with SSH1. In response, the IETF formed a working group called the Secure Shell to standardize the protocol and guide its development in the public interest. The working groups submitted the first internet draft for the SSH2 protocol in February 1997

In 1998 SCS released the software product SSH secure shell - based on the superior SSH2 protocol. However SSH2 did not replace SSH1 in the field for two reasons. First SSH2 was missing a number of useful practical features and configuration options of SSH1. Second SSH2 had a more restrictive license. The original SSH1 had been freely available from Ylonen and the Helsinki University of Technology. Newer versions of SSH1 from SCS were still freely available for most uses even in commercial settings, as long as the softwaer was not directly sold for profit or offered as a service to customers. SSH2 on the other hand was a commercial product, allowing gratis use only for qualifying education and non profit entities. As a result when SSH2 first appeared most existing SSH1 users saw few advantages to SSH2 and continued to use SSH1.SSH1 is still the most widely deployed version on the internet even though the SSH2 is a better and more secure protocol.

The situation promises to change, however as a result of two developments, a loosening of the SSH2 license and the appearance of free SSH2 implementations. Products like OpenSSH are gaining prominence. Though many people have contributed to it OpenSSH largely the work of a single software developer. It supports both SSH1 and SSH2 in a single set of programs, whereas the products provided by Ylonen SSH1 and SSH2 have separate executables. While OpenSSH was developed under OpennBSD, it has been ported successfully to Linux, Solaris and other operating systems, in tight synchronization with the main releases. Although OpenSSH is relatively new and missing some features present in SSH1 and SSH2 it is developing rapidly and promises to be a major SSH flavor in the near future.

At present time, development of SSH1 has ceased except for important bug fixes while development of SSH2 and OpenSSH remains stable and active. Other SSH implementations, notably the commercial versions of SSH1 and SSH2 maintained and sold by FSecure Corporation and numerous ports and original products for the PC Macintosh and other operating systems. It is estimated there are over two million SSH users worldwide, including hundreds of thousands of registered users of the SCS products

## Related Technologies

SSH is popular and convenient but we certainly do not claim it is the ultimate security solution for all networks, authentication, encryption and network security originated way before SSH and have been incorporated int o many other systems.

### RSB Suite

The unix programs rsb rlogin and rcp collectively known as the r commands are the direct ancestors of the SSH1 clients ssh slogin and scp. The user interfaces and visible functionality are nearly identical to

their ssh counterparts expect that ssh clients are secure. The r commands in contrast do not encrypt their connections and have are weak and easily subverted authentication model.

An r-command server relies on two mechanism for security, a network naming service and the notion of privileged TCP ports. Upon receiving a connection from a client the server obtain the network address of the originating host and translates it into a hostname. This hostname must be present in a configuration file on the server typically `/etc/hots.quiv`, for the server to permit access. The server also checks that the source TCP port number is in the range of 1-1024 since these port numbers can be used only by the UNIX superuser or root. I the connection passes both checks the server believes it is talking to a trusted program on a trusted host and logs in the client as whatever user is requested.

These two security checks are easily subverted. The translation of network address to a hostname is done by a naming service such as a DNS service. Most implementations have security holes presenting opportunities to trick the server into trusting a host it should not. Then a remote user can log into someone else's account on the server simply by having the same username.

Likewise blind trust in privileged TCP ports represents a serious security risk. A cracker who gains root privilege on a trusted machine can simply run a tailored version of the RSB client and log in as any user on the server host. Overall reliance on these port number is no longer trustworthy in a world of desktop computers, whose users have administrative access as a matter of course or whose operating systems do not support multiple users or privileges .

If user databases on trusted hosts were always synchornized with the server, installation of privileged