



Securing Remote Connections with Secure Shell

White Paper

July 2004

The Internet Protocol (IP) based communications have brought tremendous advantages and cost savings to corporations and organizations in the recent years.

While the advantages of using the Internet in everyday communications are self-evident, people are becoming more and more aware of the associated risks. IP-based networks face threats such as viruses, malicious crackers, and eavesdroppers.

Today, virus-scanning software and firewalls that prevent unauthorized access to the internal networks are widely deployed. However, this is not enough. When the Internet is used to relay confidential data, the data should be encrypted and the sender and the receiver of the data should be authenticated.

The Secure Shell (SecSh) protocol addresses these security concerns. Secure Shell is a de-facto standard for remote logins and encrypted file transfer. Secure Shell can also be used to provide tunneling for other applications, allowing the business-critical (even proprietary) applications of financial institutions, governmental organizations, and corporations to work securely over the Internet.

This document presents an overview of the Secure Shell protocol and describes the applications it enables.



INTRODUCTION

Financial institutions, governments, and large corporations rely on the Internet in their everyday business. Besides e-mail and intranets, large organizations may be running proprietary business-critical client-server applications. Examples of such applications are billing systems, access control systems, databases, and so on.

Transferring this data over the Internet brings significant cost savings. However, if the data is unprotected, malicious parties can intercept and even modify the data with little effort. Such a security breach can have disastrous consequences. Adequate security is a basic requirement when using the Internet for business.

Security Requirements

There are three core security requirements for a remote access technology.

- **Confidentiality:** The transmitted data must not be readable by unauthorized parties on the network. Confidentiality is achieved through encryption.
- **Integrity:** Unauthorized parties must not be able to modify the data without detection. Integrity is achieved by using checksum values, which reveal tampering attempts at the receiving end.
- **Authentication:** Both parties of the communication must be able to identify each other reliably, so that no one can masquerade as the other party. Authentication can be implemented by using challenge passwords, for example. However, the strongest authentication is achieved through public-key cryptography and digital signatures.

The Secure Shell protocol can provide all of these services.

THE SECURE SHELL PROTOCOL

Secure Shell (SecSh) is a de-facto standard for secure remote logins. Secure Shell secures connections over the Internet by encrypting all transmitted confidential data, including passwords, binary files, and administrative commands.

There are two versions of Secure Shell. Secure Shell version 2 (SecSh v2) provides several security improvements as compared to the original Secure Shell version 1 (SecSh v1). The protocol versions are not compatible, although some Secure Shell applications may support SecSh v1 as a fallback option. *SSH Communications Security considers Secure Shell version 1 deprecated and does not recommend its use anymore.*

Secure Shell was developed to solve the two most acute problems in the Internet, secure remote terminal logins and secure file transfers. Secure Shell can also tunnel arbitrary TCP sessions over a single encrypted Secure Shell connection. Tunneling is a powerful feature that makes it possible to secure the communication of other applications and protocols without modifying the applications themselves. By using tunnels, users can continue to use existing insecure applications, such as e-mail and X11 terminal sessions, in a secure manner. With tunneling, Secure Shell can offer an encompassing solution for securing most of the communication tasks.

Secure Shell is being standardized in the SecSh Working Group of the Internet Engineering Task Force (IETF). Secure Shell is an open and well-documented standard, and Secure Shell implementations of different



vendors have been extensively tested for interoperability.

The features of SecSh v2 are described in more detail below.

Strong Encryption

The Secure Shell version 2 protocol supports the use of the strongest available encryption algorithms, including 3DES, CAST-128, Twofish, and the new U.S. Advanced Encryption Standard (AES). The old U.S. Data Encryption Standard (DES) is also supported for interoperability reasons, but its use should be avoided, as it is no longer considered to be secure.

To give some idea of the strength of different algorithms, AES used with a 128-bit key offers protection that is likely to be unbreakable for the foreseeable future, whereas DES with its 56-bit key can be broken with special hardware almost in real-time.

Secure Shell provides data integrity by using Hash Message Authentication Codes (HMAC).

Strong Authentication

Secure Shell is a client-server protocol and both the client and the server are authenticated.

Client

The Secure Shell protocol prevents the eavesdropping of passwords allowing password-based authentication to be used safely. However, for some applications the protection offered by passwords cannot be considered secure enough. The average password contains only 28 bits of entropy, and a determined attacker may be able to crack weak passwords.

For this purpose, the Secure Shell protocol includes the possibility to use public-key authentication. Public-key authentication is based on public-key cryptography and offers strong security.

The strongest authentication and the best scalability are achieved by using an X.509-based Public Key

Infrastructure (PKI) and public-key certificates. Certificates are actually an extension of the normal public-key authentication. When certificates are used, the Secure Shell user does not have to load the public key separately to each server, but the server trusts the certification authority (CA) that has issued the certificate.

For strong, two-factor authentication, the private key used in certificate authentication can be stored on a smart card or other hardware token.

Server

To prevent man-in-the-middle attacks where a malicious party is impersonating as the server, also the server should be strongly authenticated.

In the Secure Shell protocol, the server is authenticated with a digital signature based on a DSA or RSA public-key algorithm. Each server must have a public-private key pair. In implementations without support for certificates, clients refer to a local database of trusted server public keys.

Again, the strongest and most scalable server authentication is achieved by using PKI and certificates.

USING SECURE SHELL IN YOUR ENVIRONMENT

A classic use case for Secure Shell is remote administration of a server. However, depending on the setup of your environment, Secure Shell can provide you with many additional uses to strengthen the security while being transparent to your users. These uses include secure remote command execution, forwarding insecure protocols, and advantages for a development environment.

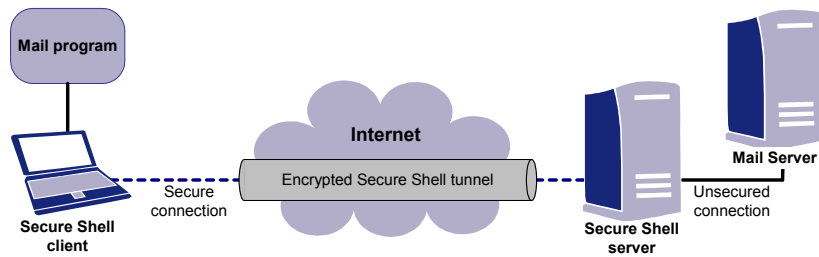


Figure 1 Example of Secure Shell usage: e-mail security through SecSh tunnelling

Secure Remote Administration

Remote Command Execution

Secure Shell has the functionality to run commands remotely through a secure connection. This enables system administrators and end users to obtain information about a remote system without having an interactive login session. For example, if the system administrator wants to find out the disk usage of one of the machines he maintains remotely, he would execute the query using Secure Shell.

Remote Backups with Secure Shell

For many system administrators today, Secure Shell has provided a valuable tool for creating secure remote backups. Secure Shell can either tunnel remote backup applications like Amanda that are TCP-based, or it can use the data dump command (dd) and be piped through Secure Shell.

Also SFTP batch runs can be used for automated transfer of backups or logs.

Programming Advantages with Secure Shell

Secure Shell can also be easily automated. It is easy to include a command-line call to `ssh2` or `sftp2` to connect to a remote server. In many cases, it is convenient to set up host-based authentication, which enables two trusted machines to connect without the need for a public key or a system password.

Business Application Protection

By using the tunneling feature, Secure Shell can be used to protect business application traffic end-to-end both in the Internet and in the corporate LAN.

Secure Shell tunneling enables system administrators to secure otherwise insecure network protocols in their network. This includes e-mail protocols, database connections that use TCP (including Oracle products), X11 applications including X-based applications, Concurrent Versions System (CVS), and even proprietary applications.

Secure Connection to the Company E-mail Server

Secure Shell can provide a simple and lightweight, yet robust, solution for secure e-mail.

Secure Connection to Proprietary Client-Server Applications

Secure Shell tunnels can provide a secure connection to business applications, for example, access to a database with financial data that needs to be encrypted even in the internal network of the company.



CONCLUSION

The Secure Shell technology provides you with network security tools that help compliment your system and data security. With Secure Shell, remote connections are encrypted and the administrators can decide which means of authentication they require. Additionally, Secure Shell enables you to create secure remote backups and tunnel other TCP-based traffic.

Using Secure Shell ensures that your mission-critical data is safe from eavesdropping while traversing the Internet and the users of the data are strongly authenticated.