



An Overview of the Secure Shell (SSH)



VANDYKE
SOFTWARE

4848 tramway ridge dr. ne
suite 101
albuquerque, nm 87111

505 - 332 -5700

www.vandyke.com

Overview of Secure Shell	2
Introduction to Secure Shell.....	2
History of Secure Shell	2
Functionality of Secure Shell.....	3
Secure Command Shell.....	3
Port forwarding	3
Secure File Transfer	4
Protocol Basics of Secure Shell	5
User Authentication	5
Host Authentication	7
Data Encryption	8
Data Integrity	8
Other Benefits	8
Secure Shell Software Solutions.....	9
VShell [®] Server	9
SecureCRT [®]	9
SecureFX [®]	9
Secure Shell – an Open Standard.....	10
Threats Addressed by Secure Shell.....	10
Eavesdropping or Password Sniffing.....	10
Man-in-the-Middle Attack (MITM)	10
Insertion and Replay Attacks	11
Need for Policy with Secure Shell	12

Overview of Secure Shell

Secure Shell (SSH) provides an open protocol for securing network communications that is less complex and expensive than hardware-based VPN solutions. Secure Shell client/server solutions provide command shell, file transfer, and data tunneling services for TCP/IP applications. SSH connections provide highly secure authentication, encryption, and data integrity to combat password theft and other security threats. VanDyke Software® clients and servers are mature native Windows implementations that offer a range of SSH capabilities and are interoperable with SSH software on other platforms.

Introduction to Secure Shell

As Internet access becomes increasingly inexpensive and available, it has become a viable replacement for traditional couriers, telephone, and fax, as well as remote dial-up access to a company's internal computer resources.

One of the biggest challenges in using the Internet to replace more traditional communications is security. In the past, companies have maintained their own modem bank dial-up access to company resources so that critical data wasn't being transmitted over the public network. Modem banks are expensive to maintain and don't scale well. In a large company, long distance charges for road warriors alone can make this an expensive solution.

Secure Shell is a protocol that provides authentication, encryption and data integrity to secure network communications. Implementations of Secure Shell offer the following capabilities: a secure command-shell, secure file transfer, and remote access to a variety of TCP/IP applications via a secure tunnel. Secure Shell client and server applications are widely available for most popular operating systems.

Secure Shell offers a good solution for the problem of securing data sent over a public network. For example, using Secure Shell and the Internet for securely transferring documents and work products electronically, rather than using a traditional overnight courier can provide a substantial cost savings. Consider that the average shipping rate for a single overnight package is between \$15 and \$30. The average one month unlimited Internet access account in the U.S. costs about \$14 a month and usually offers nationwide dial-up access. Using the Internet with Secure Shell to securely deliver your documents, you could easily recoup the cost of Internet access with just one document transfer.

History of Secure Shell

Secure Shell has seen steady improvement and increased adoption since 1995. The first version of Secure Shell (SSH1) was designed to replace the non-secure UNIX "r-commands" (rlogin, rsh, and rcp). Secure Shell version 2 (SSH2), submitted as an Internet Engineering Task Force (IETF) draft in 1997, addresses some of the more serious vulnerabilities in SSH1 and also provides an improved file transfer solution.

This increasing popularity has been fueled by the broader availability of commercially developed and supported client and server applications for Windows, UNIX and other platforms, and by the efforts of the OpenSSH project to develop an open source implementation.

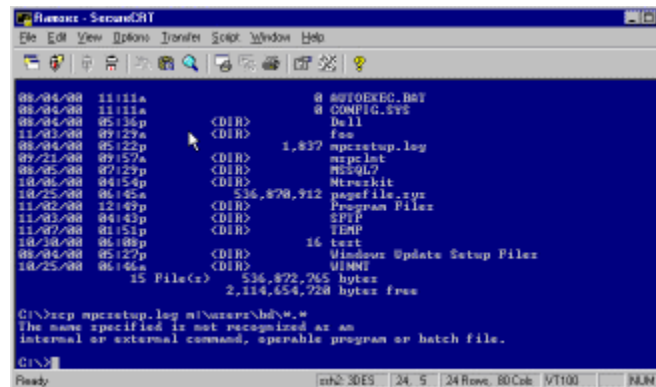
Functionality of Secure Shell

Secure Shell provides three main capabilities, which open the door for many creative secure solutions.

- Secure command-shell
- Secure file transfer
- Port forwarding

Secure Command Shell

Command shells such as those available in Linux, Unix, Windows, or the familiar DOS prompt provide the ability to execute programs and other commands, usually with character output. A secure command-shell or remote logon allows you to edit files, view the contents of directories and access custom database applications. Systems and network administrators can remotely start batch jobs, start, view or stop services and processes, create user accounts, change permissions to files and directories and more. Anything that can be accomplished at a machine's command prompt can now be done securely from the road or home.



```

08/04/08 11:11a @ AUTOEXEC.BAT
08/04/08 11:11a @ CONFIG.SYS
08/04/08 05:36p <DIR> Del1
11/02/08 09:22a <DIR>
08/04/08 05:22p <DIR> 1,837 mpsetup.log
09/21/08 09:57a <DIR> mpclnt
08/05/08 07:29p <DIR> mssql7
10/06/08 04:54p <DIR> Mircedit
10/25/08 06:45a 536,878,912 pagefile.sys
11/02/08 12:49p <DIR> Program Files
11/02/08 04:43p <DIR> SFTP
11/02/08 01:51p <DIR> TEMP
10/30/08 06:08p <DIR> 16 text
08/04/08 05:27p <DIR> Windows Update Setup Files
10/25/08 06:16a <DIR> WINNT
15 File(s) 536,872,265 bytes free
2,114,654,728 bytes free

C:\>cd mpsetup.log m\source\hd\*.
The name specified is not recognized as an
internal or external command, operable program or batch file.

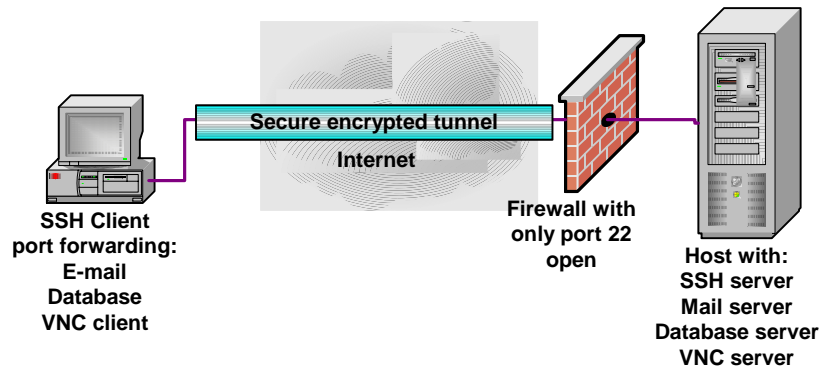
C:\>
  
```

Execute remote commands with the Secure Shell

Port forwarding

Port forwarding is a powerful tool that can provide security to TCP/IP applications including e-mail, sales and customer contact databases, and in-house applications. Port forwarding, sometimes referred to as tunneling, allows data from normally unsecured TCP/IP applications to be secured. After port forwarding has been set up, Secure Shell reroutes traffic from a program (usually a client) and sends it across the encrypted tunnel, then delivers it to a program on the other side (usually a server).. Multiple applications can transmit data over a single multiplexed channel, eliminating the need to open additional vulnerable ports on a firewall or router.

For some applications, a secure remote command shell isn't sufficient and graphical remote control is necessary. Secure Shell's port forwarding capabilities can be used to create an encrypted tunnel over which an application can be run. Virtual Network Client (VNC this will be a link to: <http://www.uk.research.att.com/vnc/index.html>), a cross platform GUI remote control application is a good example.



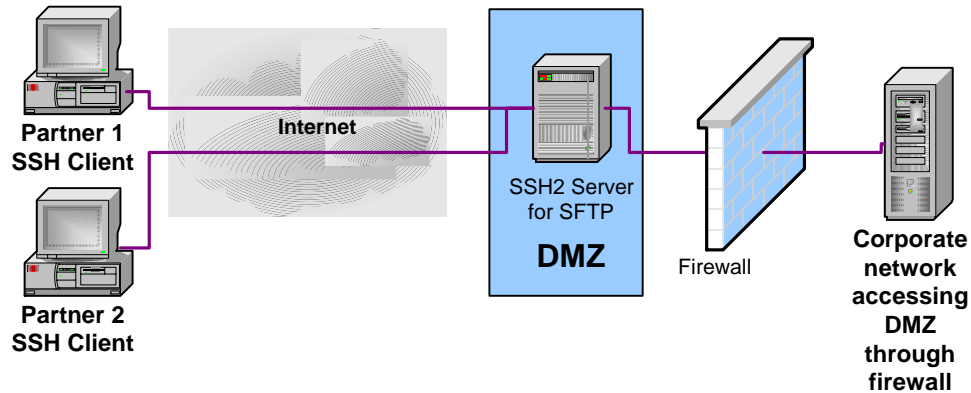
Port forwarding allows multiple TCP/IP applications to share a single secure connection

Secure File Transfer

Secure File Transfer Protocol (SFTP) is a subsystem of the Secure Shell protocol. In essence, it is a separate protocol layered over the Secure Shell protocol to handle file transfers. SFTP has several advantages over non-secure FTP. First, SFTP encrypts both the username/password and the data being transferred. Second, it uses the same port as the Secure Shell server, eliminating the need to open another port on the firewall or router. Using SFTP also avoids the network address translation (NAT) issues that can often be a problem with regular FTP. One valuable use of SFTP is to create a secure extranet or fortify a server or servers outside the firewall accessible by remote personnel and/or partners (sometimes referred to as a DMZ or secure extranet).

Using SFTP to create a secure extranet for sharing files and documents with customers and partners balances the need for access with security requirements. Typical uses of a secure extranet include uploading of files and reports, making an archive of data files available for download and providing a secure mechanism for remote administration file-oriented tasks. Extranets with business partners have proven to be much more effective for companies than more traditional methods of communication like phone or fax. In fact, SFTP can automate many of these transactions so they take place without human intervention.

A secure extranet is one of the safest ways to make specific data available to customers, partners and remote employees without exposing other critical company information to the public network. Using SFTP on your extranet machines effectively restricts access to authorized users and encrypts usernames, passwords and files sent to or from the DMZ.

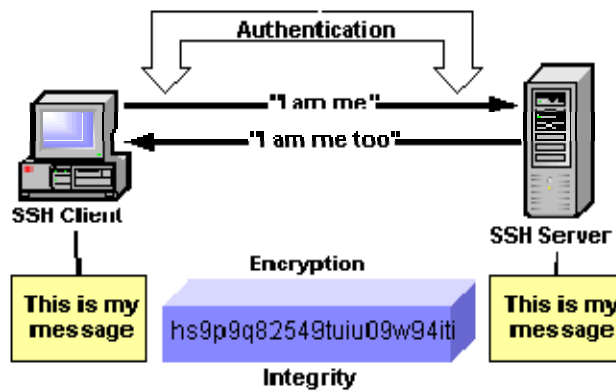


A secure extranet (DMZ) allows secure SFTP access to information assets by partners and internal users

Protocol Basics of Secure Shell

The Secure Shell protocol provides four basic security benefits:

- **User Authentication**
- **Host Authentication**
- **Data Encryption**
- **Data Integrity**



Secure Shell authentication, encryption and integrity ensure identities and keep data secure

User Authentication

Authentication, also referred to as user identity, is the means by which a system verifies that access is only given to intended users and denied to anyone else. Many authentication methods are currently used, ranging from familiar typed passwords to more robust security mechanisms. Most Secure Shell implementations include password

and public key authentication methods but others (e.g. kerberos, NTLM, and keyboard-interactive) are also available. The Secure Shell protocol's flexibility allows new authentication methods to be incorporated into the system as they become available.

Password Authentication

Passwords, in combination with a username, are a popular way to tell another computer that you are who you claim to be. If the username and password given at authentication match the username and password stored on a remote system, you are authenticated and allowed access. Some protocols like FTP and Telnet send usernames and passwords as easily visible ASCII text “in the clear”, allowing anyone with a sniffer program to easily capture them and then gain access to the system (see *Eavesdropping* for more details). Secure Shell safeguards against this attack by encrypting all data, including usernames and passwords, before transmission.

Although passwords are convenient, requiring no additional configuration or setup for your users, they are inherently vulnerable in that they can be guessed, and anyone who can guess your password can get into your system (see the *Need for policy* section for more details). Due to these vulnerabilities, it is recommended that you combine or replace password authentication with another method like public key.

Public Key Authentication

Public key authentication is one of the most secure methods to authenticate using Secure Shell. Public key authentication uses a pair of computer generated keys – one public and one private. Each key is usually between 1024 and 2048 bits in length, and appears like the sample below. Even though you can see it, it is useless unless you have the corresponding private key:

```
---- BEGIN SSH2 PUBLIC KEY ----
Subject:
Comment: my public key
AAAAB3NzaC1kc3MAAACBAKoxPsvYlv8Nu+fncH2ouLiQkuUNGIJo8iZaHdpDABAvCvLZn
jFPUN+SGPtzP9XtW++2q8khlapMUVJS0OyFWgl0ROZwZDapr2olQK+vNsUC6ZwuUDRPV
fYaqFCHrjzNBHqgmZV9qBtngYD19fGcpaqlxvHgKJFtPeQOPaG3Gt64FAAAAFQCJfkGZ
e3alvQDU8L1AVebTUFi8OwAAAIbK9ZqNG1XQizw4ValQXREczlIN946Te/1pKUZpau3W
iiDaxTF1k8Fde2714pSV3NVkWC4xlQ3x7wa6AUXIhPdLKtiUhTtxtctmllepPQS+RZKRI
XjwKL71EO7UY+b8EOAC2jBNIRtYRY0Kxsp/NQ0YYzJPfn7bqhZvWC7uiC+D+ZwAAAI EA
mx0ZY05jENA0IinXGpc6pYH18ywZ8CCI2QtPeSGP4OxxOusNdPskqBTe5wHjsZSiQr1g
b7TCmH8Tr50Zx+EJ/XGBU4XoWBJDiFp/6Bwryejo3wwjh9d4gchaozNvIXuHTCYLNPfO
RKPx3cBXHJZ27khllsjzta53BxLppfk6TtQ=
---- END SSH2 PUBLIC KEY ----
```

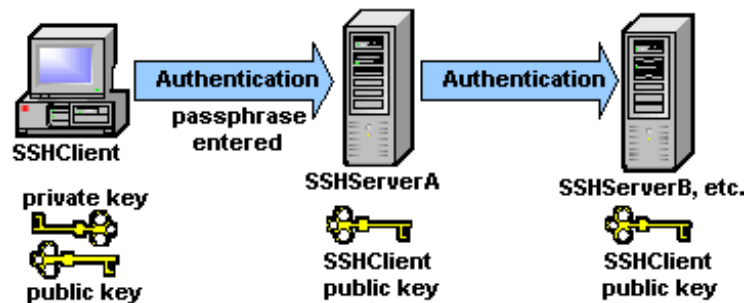
Public-private keys are typically generated using a key generation utility. Both keys in the pair are generated at the same time and, while the two are related, a private key cannot be computed from a corresponding public key. In addition to authentication, keys can also be used to sign data. To access an account on a Secure Shell server, a copy of the client's public key must be uploaded to the server. When the client connects to the server

it proves that it has the secret, or private counterpart to the public key on that server, and access is granted.

The private key never leaves the client machine, and therefore cannot be stolen or guessed like a password can. Usually the private key has a “passphrase” associated with it, so even if the private key is stolen, the attacker must still guess the passphrase in order to gain access. Public key authentication does not trust any information from a client or allow any access until the client can prove it has the “secret” private key.

Agent and Agent Forwarding

Secure Shell Agent is a way to authenticate to multiple Secure Shell servers that recognize your public key without having to re-type your passphrase each time. Additionally, by turning on agent forwarding, you can connect to a network of Secure Shell servers, eliminating the need to compromise the integrity of your private key.



Agent Forwarding passes authentication from the first SSH connection to the next, re-authenticating each time.

Notice that the private key only has to exist on the original SSHclient machine and the passphrase only needs to be typed when SSHClient connects to SSHServerA. Without agent forwarding enabled, each Secure Shell machine in the chain (except the last) would have to store a copy of the private key. SSHServerA, when authenticating SSHClient to SSHServerB becomes, in essence, a client and would require a private key to complete the authentication process. Agent support eliminates the need for the passphrase to be typed for each connection in the sequence.

Host Authentication

A host key is used by a server to prove its identity to a client and by a client to verify a “known” host. Host keys are described as persistent (they are changed infrequently) and are asymmetric—much like the public/private key pairs discussed above in the Public key section. If a machine is running only one SSH server, a single host key serves to identify both the machine and the server. If a machine is running multiple SSH servers, it may either have multiple host keys or use a single key for multiple servers. Host authentication guards against the *Man-in-the-Middle attack* (see the *Threats* section for more details). Host keys are often confused with session keys, which are used in the data encryption process discussed below.

Data Encryption

Encryption, sometimes referred to as privacy, means that your data is protected from disclosure to a would-be attacker “sniffing” or *eavesdropping* on the wire (see the *Threats* section for more details). Ciphers are the mechanism by which Secure Shell encrypts and decrypts data being sent over the wire. A block cipher is the most common form of symmetric key algorithms (e.g. DES, 3DES, Blowfish, AES, and Twofish). These operate on a fixed size block of data, use a single, secret, shared key, and generally involve multiple rounds of simple, non-linear functions. The data at this point is “encrypted” and cannot be reversed without the shared key.

When a client establishes a connection with a Secure Shell server, they must agree which cipher they will use to encrypt and decrypt data. The server generally presents a list of the ciphers it supports, and the client then selects the first cipher in its list that matches one in the server’s list.

Session keys are the “shared keys” described above and are randomly generated by both the client and the server during establishment of a connection. Both the client and host use the same session key to encrypt and decrypt data although a different key is used for the send and receive channels. Session keys are generated after host authentication is successfully performed but before user authentication so that usernames and passwords can be sent encrypted. These keys may be replaced at regular intervals (e.g., every one to two hours) during the session and are destroyed at its conclusion.

Data Integrity

Data integrity guarantees that data sent from one end of a transaction arrives unaltered at the other end. Even with Secure Shell encryption, the data being sent over the network could still be vulnerable to someone inserting unwanted data into the data stream (See *Insertion and replay attacks* for more details). Secure Shell version 2 (SSH2) uses Message Authentication Code (MAC) algorithms to greatly improve upon the original Secure Shell’s (SSH1) simple 32-bit CRC data integrity checking method.

Other Benefits

Compression, another feature of the Secure Shell protocol, is performed prior to encryption and can significantly reduce the computational cost of encrypting data. Compression can also noticeably improve the efficiency of a connection and is especially beneficial in file transfers, X11 forwarding and running curses-style programs.

Secure Shell provides helpful output or log messages. These messages can be turned on or off or configured to give varying levels of detail. Log messages can prove very helpful when troubleshooting a problem. For example, if a client were unable to connect to a given server, this log output would be the first place to look to determine the source of the problem.

Secure Shell Software Solutions

VanDyke Software provides secure solutions to vulnerable alternatives like Telnet and FTP systems. Our Secure Shell solutions, which combine the VShell™ server with the SecureCRT®, and SecureFX® clients, provide the ability to securely and remotely administer servers and routers, securely access applications, and securely transfer files. Because VanDyke Software products are based on the Secure Shell open standard, they provide customers with flexible cross-platform access while guaranteeing authentication, strong encryption, and data integrity.

VShell® Server



[VShell](#) Secure Shell server is a secure alternative to Telnet and FTP on Windows and UNIX platforms. Provide the strong encryption, robust authentication, and data integrity of SSH2 throughout your organization. Precision control over privileges, the ability to fine tune your Secure Shell environment, and a wide selection of strong authentication methods give you a flexible solution that grows with your evolving security policies.

SecureCRT®



[SecureCRT](#) is an extremely customizable terminal emulator for internet and intranet use with support for Secure Shell (SSH1 and SSH2) as well as Telnet and RLogin protocols. SecureCRT is ideal for connecting to remote systems running Windows, UNIX, and VMS. SecureCRT supports secure file transfers via Xmodem, Zmodem, and SFTP.

SecureFX®



[SecureFX](#) is a high-security file transfer client with great flexibility in configuration and transfer protocols. SecureFX includes a command-line utility for scripting batch jobs to perform secure unattended file transfers using the Secure Shell protocol (SSH). SecureFX also supports "relentless" file transfers that automatically reconnect and resume when transfer connections are broken.

Secure Shell – an Open Standard

Secure Shell is an open standard that is guided by the Internet Engineering Task Force or IETF. VanDyke Software is actively involved in the Internet Engineering Task Force standards process and has collaborated on the following contributions to the emerging Secure Shell protocol standard:

- [SECSH Public Key File Format](#)
- [GSSAPI Authentication and Key Exchange for the Secure Shell Protocol](#)

If you are interested in reading the drafts, click [here](#). The original drafts and the most recent changes may be found at <http://www.ietf.org/html.charters/secsh-charter.html> in the Internet Drafts section.

Additional Information about IETF can be found at:
<http://www.ietf.org>

Threats Addressed by Secure Shell

Below is a discussion of the threats that Secure Shell is well suited to protect your system against.

Eavesdropping or Password Sniffing

An eavesdropper is a network device, also known as a “sniffer”, which will intercept information being transmitted over the wire. This sniffing takes place without the knowledge of either the client or server and is called passive monitoring. User data including passwords can be stolen this way if you use insecure protocols like telnet and FTP. Because the data in a Secure Shell session is *encrypted*, it is not vulnerable to this kind of attack and cannot be decrypted by the eavesdropper.

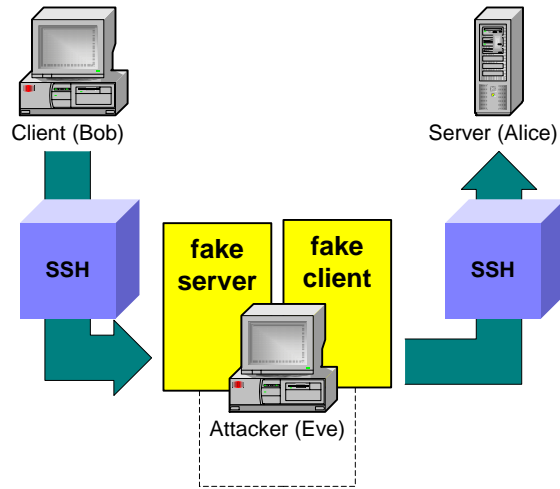
Man-in-the-Middle Attack (MITM)

If the first connection and *host key* exchange between a client and a particular host is compromised, the MITM attack fools both the client and server into thinking that they are communicating directly with one another when, in fact, an attacker is actually intercepting all traffic between the two as illustrated below:

The client (Bob) initiates a connection with the server (Alice). Unknown to both Bob and Alice, an attacker (Eve) is waiting to intercept their connection negotiation.

Eve receives Bob’s request for a connection and authenticates herself as Alice. Eve then initiates a connection with Alice posing as Bob and authenticates herself. Two secure SSH sessions are now in place with Eve reading all of the data being passed between Bob and Alice in clear text.

Secure Shell protects against MITM attacks through server [host authentication](#). Unless the host itself has been compromised, Eve does not have access to the server's private key and cannot impersonate Alice.



In a Man in the Middle attack, Eve “sits” between Bob and Alice and reads all data in the clear by impersonating Alice to Bob and Bob to Alice. Secure Shell keys prevent this attack.

Insertion and Replay Attacks

Secure Shell's implementation of [Message Authentication Code](#) algorithms prevents the threat of a “replay” or “insertion” attack. In this type of attack, the attacker is not only monitoring your Secure Shell session but is also observing your keystrokes (either physically, as in looking over your shoulder or by monitoring your terminal's keyboard with software). By comparing what you type with the traffic in the SSH stream, an attacker can deduce the packet containing a particular command (delete all files, for example) and “replay” that command at a particularly inappropriate time during your session.

Need for Policy with Secure Shell

No single piece of software can be a complete security solution. There are factors beyond securing communications through strong authentication and encryption that must be considered. The physical environment and the “human factor” are often overlooked as significant contributing factors to security breaches. The following list provides a suggested starting point for issues and areas of concern that a thorough security policy should address:

- **Password and/or passphrase** policies are needed so that users don't select short, weak or guessable passwords. In addition, you should have a policy that states how often a password should be changed, and whether or not passwords can be reused.
- **Site security** is a critical area that many organizations fail to address adequately. Portable computer users should be provided with security devices such as locking cables and encouraged not to leave these devices unattended, even for a “minute or two”. Physical access to servers, routers, network connections and backup media should be secured and limited only to those personnel who require it.
- **Security audits of service providers** are an excellent next step after your physical plant is secure and policies and procedure for your organization have been established and implemented. Internet Service Providers (ISP), Application Service Providers (ASP) and data storage vendors generally have robust physical and logical security in place. An audit may reveal deficiencies in their policies and physical plant but will more likely provide your organization with additional ideas to improve your own security plan.
- **Backup** procedures are generally adopted for servers but often overlooked or ignored for client workstations. Implementing network backup procedures can protect and insure retrieval of valuable data if a client machine is lost, stolen or damaged.

Using Secure Shell with the above policies in place will enable you to economically, privately, effectively and safely use public networks like the Internet to do your day-to-day business communications with remote users or business partners.