SSH
SSH

Set of standards and an associated network protocol that allows establishing a secure connection between a local and a remote computer.
SSH

Set of standards and an associated network protocol that allows establishing a secure connection between a local and a remote computer.

WS → Library

Firewall

Request for licensed information only available from behind the firewall

SOCKS proxy
SSH

Set of standards and an associated network protocol that allows establishing a secure connection between a local and a remote computer.

Uses public-key crypto to authenticate the remote computer and to allow the remote computer to authenticate the user.
SSH

Set of standards and an associated network protocol that allows establishing a secure connection between a local and a remote computer.

Uses public-key crypto to authenticate the remote computer and to allow the remote computer to authenticate the user.

Provides confidentiality and integrity of data exchanged between two computers using encryption and message authentication codes.
SSH

Set of standards and an associated network protocol that allows establishing a secure connection between a local and a remote computer.

Uses public-key crypto to authenticate the remote computer and to allow the remote computer to authenticate the user.

Provides confidentiality and integrity of data exchanged between two computers using encryption and message authentication codes

Uses: log into a remote machine and execute commands
tunneling
forwarding arbitrary TCP ports via SOCKS proxy
forwarding X11 connections
secure transfer of files using SFTP or SCP protocols.

Software: Daemon on the server side, Client on the user side
SSH

History:

Tatu Ylönen, Helsinki University of Technology gets pissed at eavesdroppers stealing his password – develops SSH-1 (1995)

SSH Communications Security Corp. founded in 1995.

Bugs found in SSH-1 lead to SSH-2 in 1996 – proposed as an internet standard (RFC 4251-6, January, 2006).

SSH-1 reworked by Open Source community in 1999 (original version used GPLed source, later replaced, proprietary)

Björn Grönvall's OSSH developed from this codebase.

OpenSSH shipped with the 2.6 release of OpenBSD in 1999

OpenSSH ported to numerous other OSes – this is now the most popular ssh
SSH

Key Exchange:
  Diffie-Hellman – SHA-1

Encryption:
  3des-cbc, blowfish-cbc, twofish256-cbc, twofish-cbc,
  twofish192-cbc, twofish128-cbc, aes256-cbc, aes192-cbc,
  aes128-cbc, serpent256-cbc, serpent192-cbc, serpent128-cbc,
  arcfour, idea-cbc, cast128-cbc, none, des-cbc

Message Authentication:
  hmac-sha1, hmac-sha1-96, hmac-md5, hmac-md5-96, none

Public Key:
  ssh-dss, ssh-rsa, pgp-sign-rsa, pgp-sign-dss

Compression:
  zlib, none
SSH

Secure Shell Protocol Architecture:

Four protocols:

- **Transport Layer Protocol**
  - algorithm negotiation, session key exchange, session ID, server authentication, data compression, confidentiality, transmission integrity

- **User Authentication Protocol**
  - Client authentication, public key, password or passphrase

- **Connection Protocol**
  - flow control, remote program execution, agent forwarding, port forwarding, X11 forwarding, terminal handling

- **File Transfer Protocol**
  - remote file system access, file transfers
SSH

Secure Shell Protocol Architecture:

A Secure Transport connection is established. This provides a secure, full duplex byte stream to an authenticated server.
SSH

Secure Shell Protocol Architecture:

A Secure Transport connection is established. This provides a secure, full duplex byte stream to an authenticated server.

Client sends a service request over the secure transport layer connection and is authenticated by the server.

**Note:** passwords are sent in plaintext – depend on STC
Secure Shell Protocol Architecture:

A Secure Transport connection is established. This provides a secure, full duplex byte stream to an authenticated server.

Client sends a service request over the secure transport layer connection and is authenticated by the server.

**Note:** passwords are sent in plaintext – depend on STC

Second service request sent after user authentication complete. At this point all kinds of services are available through the secure “pipe” - tunneling, SOCKS proxy, remote shell, X11 management, and so on.
SSH

Secure Shell Protocol Architecture:

A Secure Transport connection is established. This provides a secure, full duplex byte stream to an authenticated server.

Client sends a service request over the secure transport layer connection and is authenticated by the server.

**Note:** passwords are sent in plaintext – depend on STC

Second service request sent after user authentication complete. At this point all kinds of services are available through the secure “pipe” - tunneling, SOCKS proxy, remote shell, X11 management, and so on.

An application may use the File Transfer protocol under the Connection protocol established above.

**Note:** there is no security provided by SFTP, this is merely a facilitator for file transfer
SSH

Secure Shell Protocol Architecture:

A Secure Transport connection is established. This provides a secure, full duplex byte stream to an authenticated server.

Client sends a service request over the secure transport layer connection and is authenticated by the server.

**Note:** passwords are sent in plaintext – depend on STC

Second service request sent after user authentication complete. At this point all kinds of services are available through the secure “pipe” - tunneling, SOCKS proxy, remote shell, X11 management, and so on.

An application may use the File Transfer protocol under the Connection protocol established above.

**Note:** there is no security provided by SFTP, this is merely a facilitator for file transfer

**Note:** new protocols can be defined and coexist with those listed previously
SSH

Secure Shell Protocol Architecture:

Host Keys:

1. Every server host has at least one host key
   see /etc/ssh/ssh_host_ecdsa_key.pub

2. Hosts may have many keys for different algorithms

3. Multiple hosts may share the same key

4. Server host key is used during key exchange for verification

5. Thus, client must have a priori knowledge of server's p.k.

6. Trust: a) CA certs, b) local database (known-hosts)
   but b) is common due to lack of global PKI.
SSH

Account key material:

~/.ssh/known_hosts – public host keys
  gatekeeper.ececs.uc.edu,129.137.4.128 ssh-rsa AAAAB3Nz...

~/.ssh/authorized_keys – keys from other hosts
  ssh-rsa AAAAB3NzaC... oJkRC6pw== franco@home.cinci.rr.com

~/.ssh/id_rsa.pub – user's public key
  ssh-rsa AAAAB3NzaC1yc2... LQnN+AQ== franco@franco

~/.ssh/id_rsa – user's private key
  -----BEGIN RSA PRIVATE KEY-----
  Proc-Type: 4,ENCRYPTED
  DEK-Info: DES-EDE3-CBC,FF76B74E262ECC85

  Fszuf4IibMJNWUz8UODS05QwvmeT5JbAy8FLkCXDQhfK7Mxp9ybkakGW7SGk7bxQ...
  G8+Tzj/Q/kqrejhY77q5EREGUXs1IuwwCL83UzRJRysY0Nx4i79mw==
  -----END RSA PRIVATE KEY-----

~/.ssh/environment – add some things to env
  DO_NOT_UMOUNT=1
**SSH**

**Client Configuration:**

```
/etc/ssh/ssh_config

ForwardAgent yes
PasswordAuthentication yes # allow password authentication
CheckHostIP yes # check host IP also
Tunnel no # device forwarding: P-P, eth
TunnelDevice any:any # device to open
ForwardX11 yes # X11 redirected to channel
ForwardX11Trusted yes # trust remote X11 clients
PermitLocalCommand no # execute command after connect
VisualHostKey no # print key pattern at login
AddressFamily any # IPv4 or IPv6
BatchMode no # disable pass ask – no user
 # is present to respond
GSSAPIAuthentication yes # Generic Security Services
GSSAPIDelegateCredentials no # forward GSS credentials
GSSAPIKeyExchange no #
```
SSH

Server Configuration:

/etc/ssh/sshd_config

PubKeyAuthentication yes
PasswordAuthentication yes
PermitEmptyPasswords no
KerberosAuthentication yes
KerberosOrLocalPasswd yes
KerberosTicketCleanup yes
KerberosGetAFSToken no  # Andrew File System services
GSSAPIAuthentication no  # Generic Security Services
GSSAPICleanupCredentials yes # destroy user credentials
AllowAgentForwarding yes
TCPKeepAlive yes  # check connection is alive
UsePrivilegeSeparation yes # create unprivileged child
# process to deal with network
# traffic. Ask privileged
# parent to do privileged tasks
# Prevents privilege escalation
# via corruption due to inet
PermitUserEnvironment no  # allow creation of user
# environment
SSH

Secure Shell Transport Layer Protocol:
Provides encryption, host authentication, integrity protection.
1. Client authentication is not performed
2. All crypto algorithms and parameters are negotiated
3. Key Exchange:
   - each side sends name-lists of supported algorithms
   - after agreement, key exchange takes place
   - result is shared secret $K$ and exchange hash $H$
   - $K$ and $H$ used to derive encrypt, authen keys (6 hashes)
   - $H$ is used as a session identifier
SSH

Secure Shell Transport Layer Protocol:

Key Exchange:

Client: `SSH2_MSG_KEXINIT` ; cookie (random 16 bytes)
identification string (e.g. `SSH-2.0-billsSSH_3.6.3q3<cr><lf>`)  
Client: `diffie-hellman-group-exchange-sha256,diffie-hellman-group-exchange-sha1,diffie-hellman-group14-sha1,diffie-hellman-group1-sha1` (key exchange algorithms)
Client: `ssh-rsa,ssh-dss` (host key algorithms)
Client: `aes128-ctr,aes192-ctr,aes256-ctr,arcfour256,arcfour128,aes128-cbc,3des-cbc,blowfish-cbc,cast128-cbc,aes192-cbc,aes256-cbc,arcfour,rijndael-cbc@lysator.liu.se` (encryption algorithms)
Client: `hmac-md5,hmac-shal,umac-64@openssh.com,hmac-ripemd160,hmacripemd160@openssh.com,hmac-shal-96,hmac-md5-96,hmac-md5,hmac-shal,umac-64@openssh.com,hmac-ripemd160,hmac-ripemd160@openssh.com,hmac-shal-96,Hmac-md5-96` (integrity algorithms)
Secure Shell Transport Layer Protocol:

Key Exchange:

Client: none, zlib@openssh.com (data compression algorithms)
Server: similar to previous client transmissions
Client: server->client aes128-ctr hmac-md5 none (choice)
Server: client->server aes128-ctr hmac-md5 none (choice)
Client: dh_gen_key: request p and g of max, min bit sizes
Server: send p and g to client
Client: SSH2_MSG_KEX_DH_GEX_INIT sent (DH half sent: \( e = g^a \mod p \))
SSH

Secure Shell Transport Layer Protocol:

Key Exchange:

Client: none,zlib@openssh.com (data compression algorithms)
Server: similar to previous client transmissions
Client: server->client aes128-ctr hmac-md5 none (choice)
Server: client->server aes128-ctr hmac-md5 none (choice)
Client: dh_gen_key: request p and g of max, min bit sizes
Server: send p and g to client

Client: SSH2_MSG_KEX_DH_GEX_INIT sent (DH half sent: \(e=g^a \mod p\))
Server: computes its DH half \((f = g^b \mod p)\) and key \(K = e^b \mod p\)
computes \(H = Hash(ID_C + ID_S + IN_C + IN_S + K_S + e + f + K)\)
where \(ID_X\) = identification string of X
\(IN_X\) = SSH_MSG_KEXINIT message of X
\(K_S\) = public host key of server
sends \(f, K_S\), signed \(H\) (with its private host key) to client
SSH

Secure Shell Transport Layer Protocol:

Key Exchange:

Client: already knows ID_S, ID_C, IN_S, IN_C, e, p

gets $f$ from server and computes $K = f^a \mod p$

gets $K_S$ from server and computes

$$H = \text{Hash}(ID_C + ID_S + IN_C + IN_S + K_S + e + f + K)$$

gets the signed $H$ from server and decrypts with $K_S$

compares result with $H$ to verify $K_S$

compares $K_S$ with the server's hostname in the “known-hosts”
file – looks at IP address & hostname!

Server has been authenticated.
Secure Shell Transport Layer Protocol:

**Key Exchange:**

Client and Server compute keys:

- Initial IV client to server: Hash(K + H + "A" + session_id)
- Initial IV server to client: Hash(K + H + "B" + session_id)
- Encryption key client to server: Hash(K + H + "C" + session_id)
- Encryption key server to client: Hash(K + H + "D" + session_id)
- Integrity key client to server: Hash(K + H + "E" + session_id)
- Integrity key server to client: Hash(K + H + "F" + session_id)

Secure Transport Connection is Established
SSH

Transport Packet Formation

**Seq #**: helps prevent replay attack – only with MAC

**Pad**: total length of packet less MAC is multiple of cipher block size

```
payload
  ↓
  Compress
  ↓
  Compressed Payload
  ↓
pktl  pdl  Padding
  ↓
  Encrypt  MAC
  ↓
ciphertext
```

pktl = Packet Length
pdl = Padding Length
**SSH**

**Replay**

The cookie and sequence # provide a unique identifier that can be used by higher level protocols to bind data to a given session and prevent replay of data from prior sessions.

**Man-in-the-middle**

This protocol makes no assumptions or provisions for an infrastructure or means for distributing the public keys of hosts. It is expected that this protocol will sometimes be used without first verifying the association between the server host key and the server host name.

**Denial-of-service**

Vulnerable to Denial of Service attacks because an attacker can force the server to go through the CPU and memory intensive tasks of connection setup and key exchange without authenticating.

**Covert Channels**

Protocol was not designed to eliminate covert channels – bad guys may establish communication between accounts that should not be talking.
SSH

Secure Shell Authentication Protocol:
General-purpose user authentication protocol. Provides the means by which the client is authenticated. Intended to be run over the SSH transport layer protocol, assumes underlying protocols provide integrity, confidentiality.

- Server gives client available authentication methods
- Client has freedom to try methods listed in any order. (Server controls authentication but client has flexibility to use methods it has)
- The only required method of authentication is via DSS public key
SSH

Secure Shell Authentication Protocol:

**Client:** SSH_MSG_USERAUTH_REQUEST none

**Server:** user-auth request for user franco service ssh-connection
method none
failed none for franco 10.63.1.250 port 50495 ssh2
Authentications that can continue: publickey,
gssapi-keyex,gssapi-with-mic,password

**Client:** preferred gssapi-with-mic,publickey,keyboard-interactive,password
(try general security services API first)

**Server:** authmethod_lookup gssapi-with-mic
remaining preferred: publickey,keyboard-interactive,password
authmethod_is_enabled gssapi-with-mic
Next authentication method: gssapi-with-mic
Trying to reverse map address 10.63.1.250.
Unspecified GSS failure.
Credentials cache file '/tmp/krb5cc_500' not found
we did not send a packet, disable method
Next authentication method: publickey  (try RSA next)
SSH

Secure Shell Authentication Protocol:

**Client:** Offering public key: `/home/franco/.ssh/id_rsa`

**Server:** Server accepts key: `pkalg ssh-rsa blen 277`
  
  `input_userauth_pk_ok: SHA1 fp 4a:c4:e5:39:08:f3:de:...`
  
  (authorization request – look in `/home/franco/.ssh/authorized_keys`)

**Client:** `sign_and_send_pubkey` (authentication request)

**Server:** Authentication succeeded (publickey).
**SSH**

**Secure Shell Connection Protocol:**

Runs on top of the transport layer and authentication protocols.

Provides interactive login sessions, remote execution of commands, forwarded TCP/IP connections, and forwarded X11 connections.

- Multiple service requests managed through 'channels'

**Channel types:**

- **session**: remote execution of a program.
  could be an application, shell, system command
- **X11**: allows GUI applications to be run on remote machine
  with results appearing on local machine
- **forwarded-TCP/IP**: remote port forwarding (tunneling)
  `ssh -R 20000:localhost:20000 helios.com`
- **direct-TCP/IP**: local port forwarding
  `ssh -L 8150:helios.com:8150 helios.com`
**SSH**

**Secure Shell Connection Protocol:**

**Client:** `ssh helios.ececs.uc.edu ls`  
Client: channel 0: new [client-session]  
channel 0: send open  
Requesting no-more-sessions@openssh.com  
Entering interactive session.  
Requesting X11 forwarding with authentication spoofing.  
channel 0: request x11-req confirm 0  
Requesting authentication agent forwarding.  
channel 0: request auth-agent-req@openssh.com confirm 0

**Server:** client_session2_setup: id 0

**Client:** Sending environment.

**Client:** Sending command ls.  
Command is executed

**Server:** channel 0: gc: user detached  
channel 0: send close  
channel 0: is dead  
channel 0: garbage collecting  
channel 0: free: client-session, nchannels 1
SSH

Secure Shell Connection Protocol:


Client: Local connections to LOCALHOST:2000 forwarded to remote address gauss.ececs.uc.edu:2000

channel_setup_fwd_listener: type 2 wildcard 0 addr NULL
Local forwarding listening on ::1 port 2000.
channel 0: new [port listener]
Local forwarding listening on 127.0.0.1 port 2000.
channel 1: new [port listener]
channel 2: new [client-session]

ssh_session2_open: channel_new: 2
channel 2: send open
Requesting no-more-sessions@openssh.com
**SSH**

**Examples:**

sftp franco@gauss.ececs.uc.edu

ssh -N -f -L 8180:gauss.ececs.uc.edu:8180 helios.ececs.uc.edu

scp gauss.ececs.uc.edu:14.ps helios.ececs.uc.edu:

ssh -D 8000 helios.ececs.uc.edu

ssh gauss "tar cf - ./*.pdf" | (cd /tmp ; tar xpvf -)

rsync -ve ssh franco@gauss:/tmp/* .pdf

rsync -auvp ~/[A-Z]* franco@boole.ececs.uc.edu: