SSH Transport Layer Protocol

Tatu Ylönen
SSH Communications Security
e-mail: ylo@ssh.fi

Overview

- Strong encryption, integrity protection, and mutual authentication
- Can interface to almost any public key infrastructure
- All algorithms, public key types, and key exchange methods are negotiated
- Typical one round-trip total (two worst case) for algorithm negotiation, key exchange, mutual authentication, and service request
- Supports key re-negotiation (also for pre-encrypted/pre-authenticated data)
- Can replace Unix `inetd` functionality for secure services

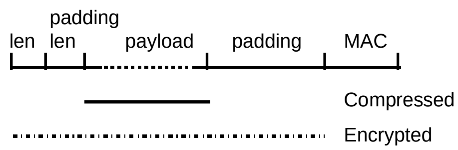
Connection Setup

- Can run over any binary-transparent transport
- Both sides send a protocol version without waiting for the other side
- TCP/IP use: server listens for connections on port 22. After key exchange and authentication, client requests a named service, and server spawns an application server. Concept similar to unix inetd.

```
finger   nobody  /usr/sbin/fingerd  fingerd
sshlogin root    /usr/sbin/sshd     sshlogind
sftp     root    /usr/sbin/sftp     sftpd
http     nobody  /usr/sbin/shttp    shttpd
```

- Little/no change in servers that communicate using stdin/stdout
- Could be implemented in standard socket libraries on client

Binary Packet Protocol



- Handles compression, encryption, data integrity
- Compression: e.g. GZIP
- Encryption: e.g. IDEA-CBC, 3DES-CBC, Blowfish-CBC, RC4
- MAC: e.g. HMAC-MD5, HMAC-SHA

Algorithm Negotiation

- Both sides send SSH_MSG_KEXINIT with cookie, kex_algorithms list, host_key_algorithms list, public_key_algorithms list, encryption_algorithms lists for each direction, mac_algorithms lists for each direction, compression_algorithms lists for each direction, hash_algorithms list
- Chosen algorithm is first one listed by client that is also supported by server
- Each side makes a guess of the algorithm choice, and may follow this packet immediately with the first (guessed) key exchange packet

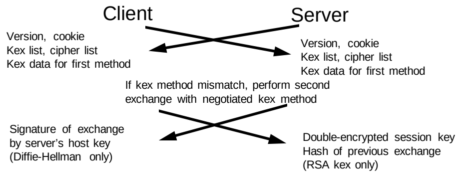
RSA-Style Key Exchange

- Client uses encryption-capable host key and short-lived server key to encrypt random session key:
  - » 1. Server sends public host key and public server key (server key changes typically every hour). Client validates server host key from public key infrastructure or local database.
  - » 2. Client generates random shared secret, computes SHA hash of key exchange so far, encrypts both, first with server key and then with session key. Sends the resulting value to the server.
  - » 3. Server decrypts session key and hash using its private host key and private server key, and verifies hash against its idea of the key exchange.
- Ability to decrypt session key acts as response to challenge.
- Any encryption-capable public key method can be used.

Diffie-Hellman Key Exchange

- Diffie-Hellman exchange followed by signature exchange
- Any Diffie-Hellman style operation can be used (e.g. elliptic curve variants). Any signature-capable host key can be used.
- Details not yet specified.

Key Exchange



Key Derivation

- Key exchange derives a 256-bit shared secret. Keys are derived by computing hash of cookies and a portion of the shared secret as follows:
  - Initial IV client to server: bytes 0-15
  - Initial IV server to client: bytes 1-16
  - Encryption key client to server: bytes 5-20
  - Encryption key server to client: bytes 8-23
  - Integrity key client to server: bytes 13-28
  - Integrity key server to client: bytes 16-31
- Encryption, compression, and MAC are enabled immediately after key exchange.

Client Host Authentication

- Client sends its name, public host key, and signature of host name + cookies. Server validates host key using public key infrastructure or local database.
- Client host authentication is optional.
- Client may send multiple authentications with different host key types if there is doubt whether the server is able to validate a particular type of host key. (Multiple different host keys are supported.)

Service Request

- After key exchange and host authentication, the client requests a particular service. Services are identified by a name. It is expected that there will be an `inetd.conf` -like configurable table of available services.
- The server may communicate with application servers using either a transparent byte stream or a packet stream.

Data Exchange

- SSH_MSG_DATA passes data in either direction. Maximum packet size is 35.000 bytes.
- SSH_MSG_EOF indicates end of input.
- SSH_MSG_CLOSE closes connection.
- The SSH server passes data to application server as either a byte stream or as packets, depending on the service.

Key Renegotiation

- Either side may request renegotiation at any time. Renegotiation is recommended after every gigabyte of transferred data or one hour of connection time.
- Renegotiation is basically a new key exchange, performed under whatever encryption is currently in effect.
- Algorithms and even host keys may change during renegotiation. (Host key change may be important to avoid disconnecting long-lived sessions when a host key expires.)
- Renegotiation uses public key techniques, which makes renegotiated keys fully independent from previous keys. Can safely switch to no encryption/no MAC for pre-encrypted/pre-authenticated data if desired.
- Not very lightweight due to public key operations.

Public Key Infrastructure

- Keys, signatures, and encryption formats negotiated and passed as binary blobs. Almost any public key algorithm and format can be used, and many can be supported simultaneously.
- Permits multiple host keys with different algorithms for the same host.
- Currently no interface to an infrastructure is defined since no widely used infrastructure exists yet. It is expected that one will be defined in the next few months.
- Currently a simple RSA-PKCS1 key format is used, but new formats (including X.509, if desired) can be defined in future.

More Information

- Technical issues and the protocol specification
  - » http://www.ssh.fi/
  - » e-mail: Tatu Ylönen <ylo@ssh.fi>
- Commercial implementation
  - » http://www.datafellows.com/
  - » e-mail: f-secure-ssh-sales@datafellows.com