SSH - Secure SHell

Lecture 23

Slides prepared by Joseph Zhaojun Wu
Revised by Cunsheng Ding in 2025



Outline

- Brief introduction
- Protocol details
  - The official documentation of SSH has over 100 pages.
  - Hence, our presentation is at a high level.
- Applications
- References



Brief Introduction



What is SSH?

- It is a set of standards and associated protocols to establish a secure channel between two computers.
- It provides mutual authentication, data confidentiality, and data integrity.
- Originally, it was designed as a replacement of insecure applications like r-commands (i.e., Berkeley remote commands, e.g., rlogin, rsh, rcp).



Motivations of Designing the SSH

- The following is a list of drawbacks in some *traditional* applications:
  - Authentication is based on IP address
  - Authentication is based on reusable password
  - Data is transmitted in clear text
  - X protocol is vulnerable to attack
  - Intermediate hosts can hijack sessions

For X protocol, see http://en.wikipedia.org/wiki/X_Window_System



Applications of SSH

- Secure remote login (ssh client)
- Secure remote command execution
- Secure file transfer and backup (sftp/rsync/scp)
- Public-private key pair generation for you and an agent for taking care of your public-private key pair
- Port forwarding and tunnelling (x11 forwarding and tunnelling using SSH)



Brief History



- Tatu Ylönen, a researcher at Helsinki University of Technology, Finland, developed the first version of SSH in 1995.
- Very popular, 20K users in 50 countries in the first year.
- Ylönen found the SSH Communications Security (www.ssh.com) to maintain, develop and commercialize SSH, in Dec. 1995.
- SSH2 was released in 1998.

Brief History (cont.)

- 1999, Björn Grönvall developed the OSSH.
- "OpenBSD" then extended Grönvall's work,and launched the OpenSSH project (www.openssh.org),
- OpenSSH was ported to Linux, Solaris, AIX, Mac OS X, Windows (cygwin) and etc.
- Currently, OpenSSH is the single most popular SSH implementation in most operating systems.

Remark: The OpenBSD project produces a **FREE**, multi-platform UNIX-like operating system.



SSH Implementations

Name	UNIX	WIN	MAC	Clients	Server	FREE
SSH.COM	X	X		X	X	
OpenSSH	X	X		X	X	X
F-Secure SSH	X	X	X	X	X	
PuTTY		X		X		X
SecureCRT, SecureFX		X		X		
VShell		X			X	
TeraTerm		X		X		X
MindTerm	X	X	X	X		X
MacSSH			X	X		X



IPSec & SSL vs. SSH

- IPSec is a lower level (IP-based) security solution than SSH. More fundamental but really expensive. SSH is quicker and easier to deploy.
- SSL or TLS is TCP-based and "mainly" used in WEB applications.
- There are some SSL-enhanced Telnet/FTP applications. SSH is a more integrated toolkit designed just for security.



Protocol Details



SSH Architecture

- SSH protocol is based on a *client/server* architecture
  - A ssh server running on the server side is listening on the 22 TCP port for incoming connection

```
joseph@hlt029:~> sudo netstat --tcp --listening --program
tcp6  0  0 *:ssh *:*        LISTEN     3075/sshd
```

  - A client who wants to connect to a remote host will execute the *ssh* command

```
joseph@PeT43:~> ssh hlt029
```

Remark: Port 22/TCP,UDP: for SSH (Secure Shell) - used for secure logins, file transfers (scp, sftp) and port forwarding



Building Blocks

SSH-2 Protocol has the following three building blocks (RFC 4251, 29 pages):

- Transport Layer (RFC 4253, 31 pages):

  Initial key exchange, *server* authentication, data confidentiality, data integrity, compression (optional), and key re-exchange.

- User Authentication Layer (RFC 4252, 16 pages):

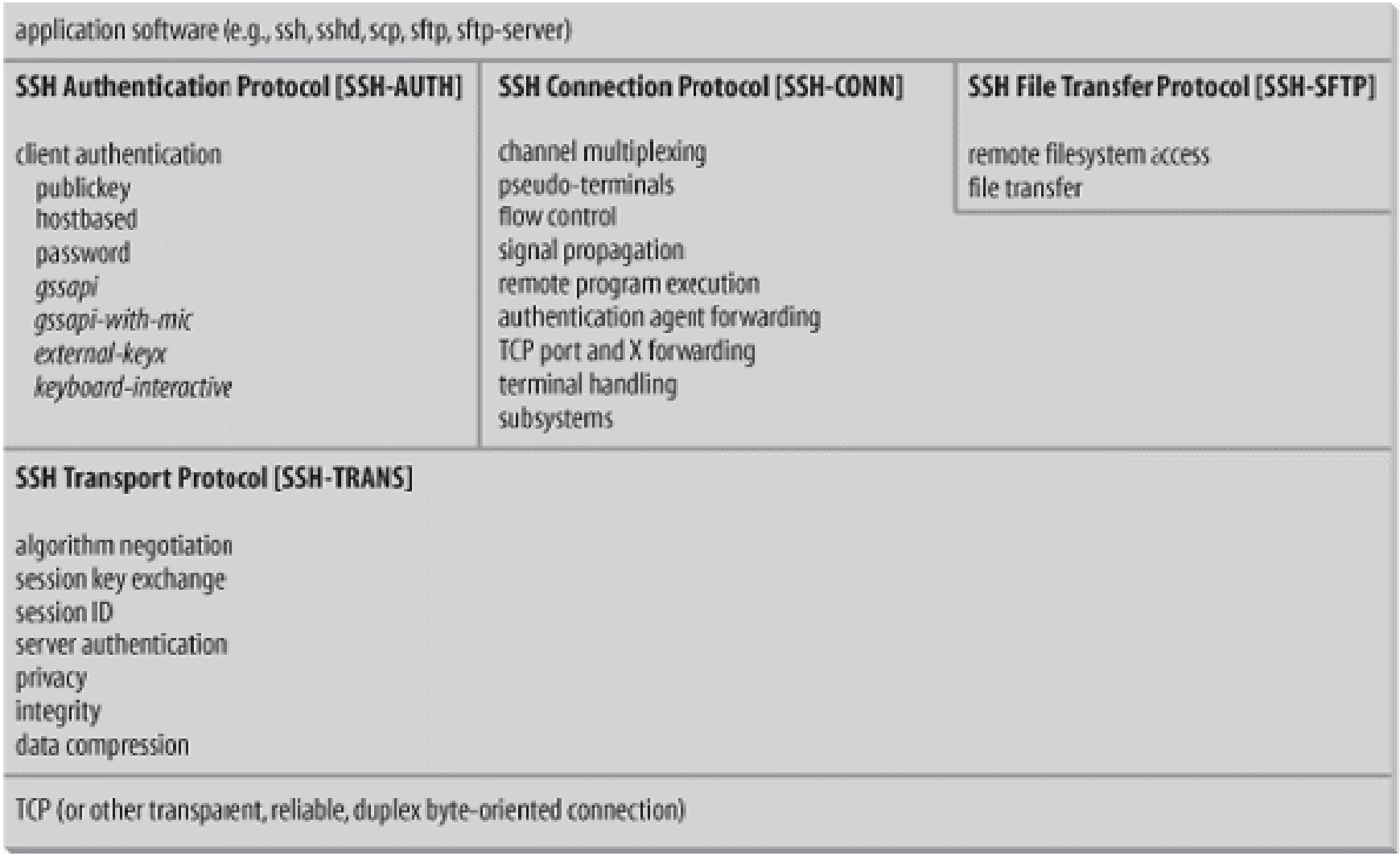
  Client authentication, provide various authentication methods.

- Connection Layer (RFC 4254, 28 pages):

  Defines the logical *channels* and the *requests* to handle the services like: secure interactive shell session, TCP port forwarding and X11 forwarding.



Structure of the Building Blocks



Outline

- Protocol Details
  - Transport Layer
  - User Authentication Layer
  - Connection Layer



Transport Layer

- It is a fundamental building block of SSH.
- It provides services such as the initial connection, server authentication, data encryption and data integrity.
- It is used for the negotiation of cryptographic algorithms.



Parameters Negotiated by the Transport Layer

- A key exchange algorithm in the form diffie-hellman-group-exchange-sha1 for computing a master key and session ID
- A list of client authentication methods and a server authentication method.
- Two data encryption ciphers for encrypting data in the two directions.
- Two data integrity algorithms in the form: hmac-hashfunction.
- Two data compression algorithms for compressing data in the two directions (optional).

Parameter negotiation details are omitted here (RFC 4253).



Key Exchange & Server Auth.

- After the parameter negotiation, the DH key exchange protocol is carried out and a master key is computed.
- Server authentication is done with the public key of server
  - In other words, server authentication is based on the server's digital signature.
  - Server authentication will be discussed later.



Computing other Keys

- Based on the shared master key, both sides compute the data encryption keys and data integrity keys (details are omitted)
  - The two encryption keys are independent
  - The two data integrity keys (i.e., authentication keys) are independent
- After finishing the server authentication and the key exchange, the client has a single, secure, full duplex stream to an authenticated server



Remarks on the Parameter Negotiation

- The Key Exchange produces two values:
  - a shared secret K (master key) and an exchange hash value H (details are omitted).
- The unique H is used as the Session ID.
- Data flow directions client->server and server->client are independent, may use different algorithms (i.e. 3DES+SHA1 and Blowfish+MD5)
  - But in practice, it is **recommended** that the same cipher and same hash function are used for both directions.
- If compression is enabled, the data is first compressed and then encrypted.



Server Authentication (continued)

- It is done by verifying the server's digital signature with the RSA or DSS public key of the server.
- If a client is contacting a server with SSH the first time, the client needs the public key of the server for server authentication.
- How does the client get the public key of the server the first time???



How to get server's public key the 1st time?

- Two trusted methods:
  - the client maintains a local database that associates each server name and the corresponding public key.
  - The client gets the public key of a server from a trusted 3rd party (e.g. Certification Authority)
  - These two methods are rarely used.
- Another Option: server authenticity is not checked the first time
  - After the first connection, the public key of the server is saved in a database of the client.
  - This is the most used method.



Required/Recommended Algorithms

- Key Exchange:
  - diffie-hellman-group1-sha1 [Required]
  - diffie-hellman-group14-sha1 [Required]
- Data Encryption:
  - 3des-cbc [Required]
  - aes128-cbc [Recommended]
- Data Integrity:
  - hmac-sha1 [Required],
  - hmac-sha1-96 [Recommended]
- Public Key for Authentication:
  - ssh-dss [Required]
  - ssh-rsa [Recommended]



Outline

- Protocol Details
  - Transport Layer
  - User Authentication Layer
  - Connection Layer



User Authentication Layer (1)

- It runs atop of the Transport Layer
- It relies on the data privacy and integrity, provided by the Transport Layer
- Service ID: "ssh-userauth"
- Several user authentication methods are available



User Authentication Layer (2)

- Client requests the service "ssh-userauth"
- Server responds with the list of available user authentication methods. More than one user authentication methods may be required.
- Methods:
  - Public key [Required]
  - Password
  - Host-based



User Authent. Request & Server Response

- User Authentication Request is driven by the client and has the following parts:
  - user name
  - service name
  - method name

- Authentication Response from the server:
  - SUCCESS: user authentication done.
  - FAILURE: return a list of user authentication methods that can continue



Outline

- Protocol Details
  - Transport Layer
  - User Authentication Layer
  - Connection Layer



Connection Layer

- It runs over the Transport Layer, and utilizes the User Authentication Layer
- It multiplexes the encrypted connection provided by the Transport Layer into several logical channels
- Channel type:
  - Interactive session
  - Remote command execution
  - An X11 client connection
  - TCP/IP port forwarding
  - …

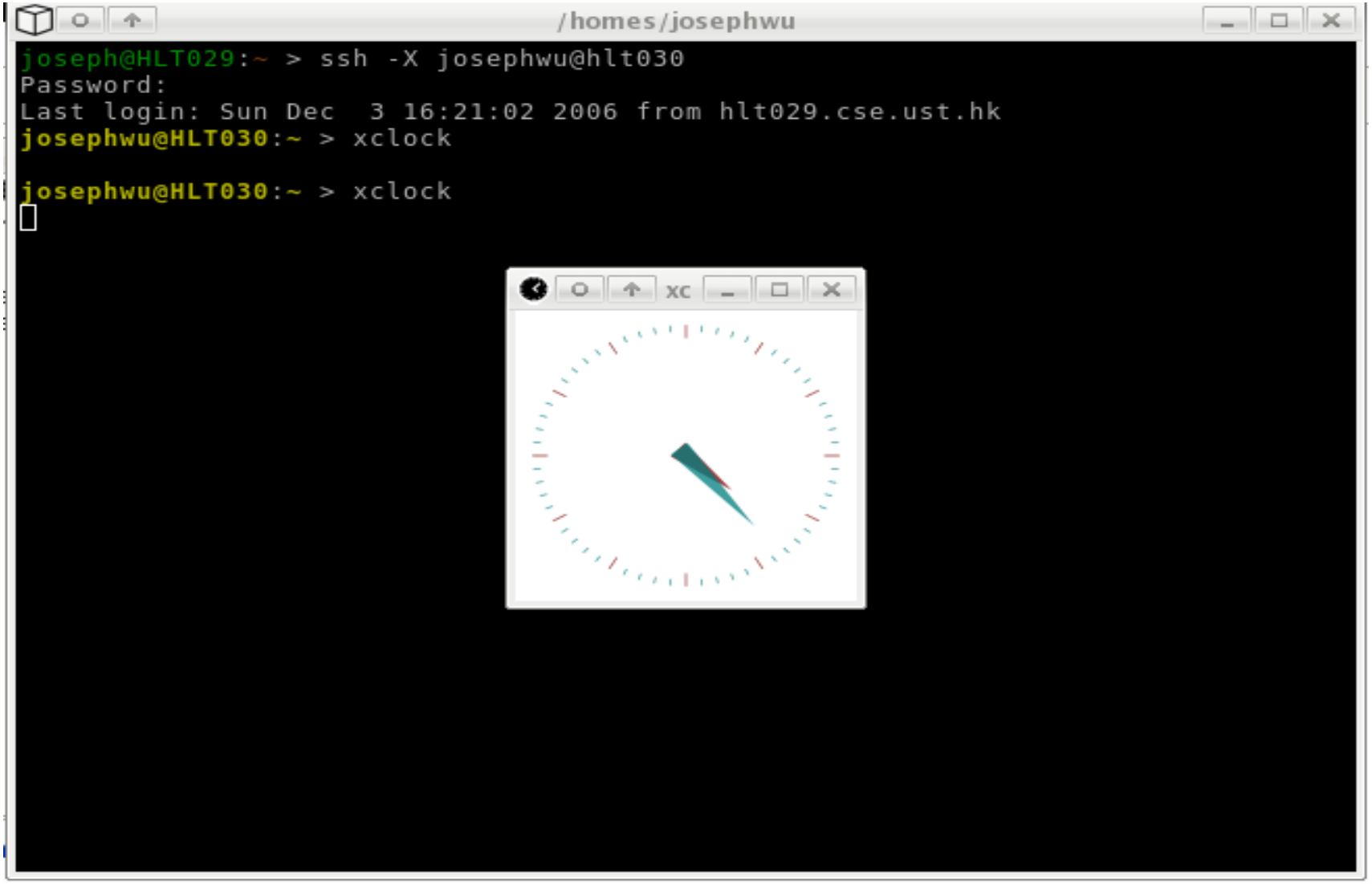
Please stop and explain the meaning of “multiplexing a connection” using a physical tunnel through a mountain in HK



Applications of SSH



X11 Forwarding

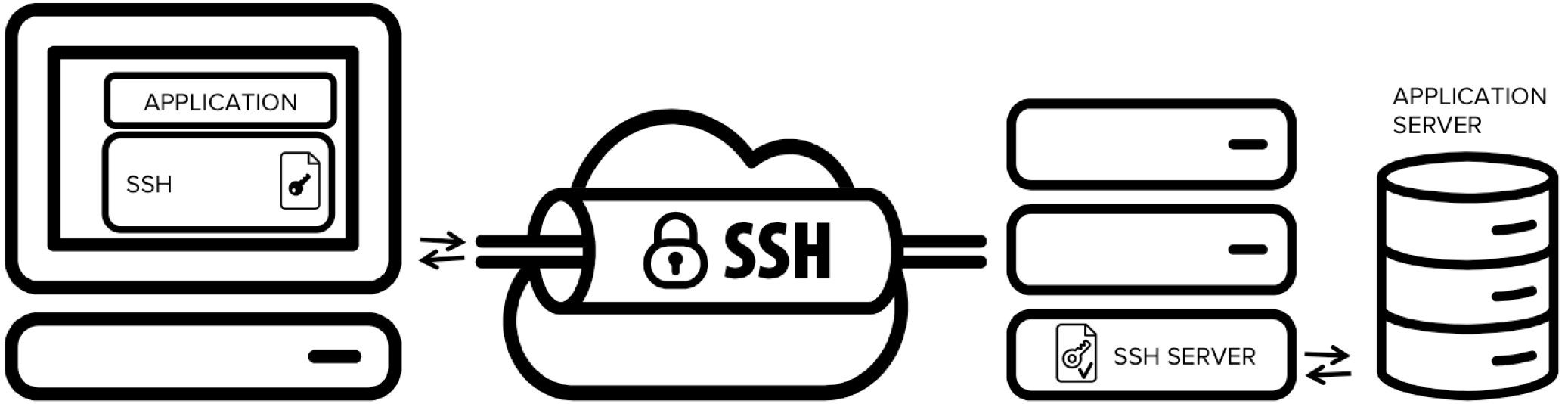


Port Forwarding

- SSH port forwarding is a mechanism in SSH for tunneling application ports from the client machine to the server machine, or vice versa.
- The application data traffic is directed to flow inside an encrypted SSH connection so that it cannot be eavesdropped or intercepted while it is in transit.
- SSH tunneling adds network security to legacy applications that do not natively support encryption.



Port Forwarding (Continued)



SCP, and SFTP

- SCP: copying files between hosts by using SSH for data transfer.
- SFTP: Secure FTP over SSH.



SSH Public Key Authentication

- Using "*ssh-keygen*" you can ask SSH to generate an RSA or DSA key pair for you, with protection of your private key from a passphrase:
- SSH will then add your public key into the server's "authorized_keys" database. (~/.ssh/authorized_keys)
- Later you can connect to the server by authenticating yourself with your public key.



References

- SSH: The Secure Shell The Definitive Guide 2E
- SSH FAQ
- OPENSSH Project Official Site
- SSH Communications Security
- The Secure Shell (SSH) Protocol Architechture
  https://datatracker.ietf.org/doc/html/rfc4251
- The SSH Transport Layer Protocol
  https://www.rfc-editor.org/rfc/rfc4253#section-7.1

