

# SSH - Secure SHell

10101101010100010110101110101010100010010101000110101101010100010110

# Outline

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

101011010100010110101110101010100010010101000110101101010100010110

## 3

# 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).

10101101010001011010111010101010001001010001101011010100010110

# 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](http://en.wikipedia.org/wiki/X_Window_System)

10101101010001011010111010101010001001010001101011010100010110

# 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)

101011010100010110101110101010100010010101000110101101010100010110

# 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](http://www.ssh.com)) to maintain, develop and [commercialize](http://www.ssh.com) SSH, in Dec. 1995.
- SSH2 was released in 1998.

1010110101000101101011101010101000100101000110101101010100010110

# Brief History (cont.)

- 1999, Björn Grönvall developed the OSSH.
- “OpenBSD” then extended Grönvall's work, and launched the OpenSSH project ([www.openssh.org](http://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.

10101101010001011010111010101010100010010101000110101101010100010110



# 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

1010110101000101101011101010101000100101010001101011010100010110

# 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.**

10101101010100010110101110101010100010010101000110101000110101101010100010110

## 11

# 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

101011010100010110101110101010100010010101000110101101010100010110

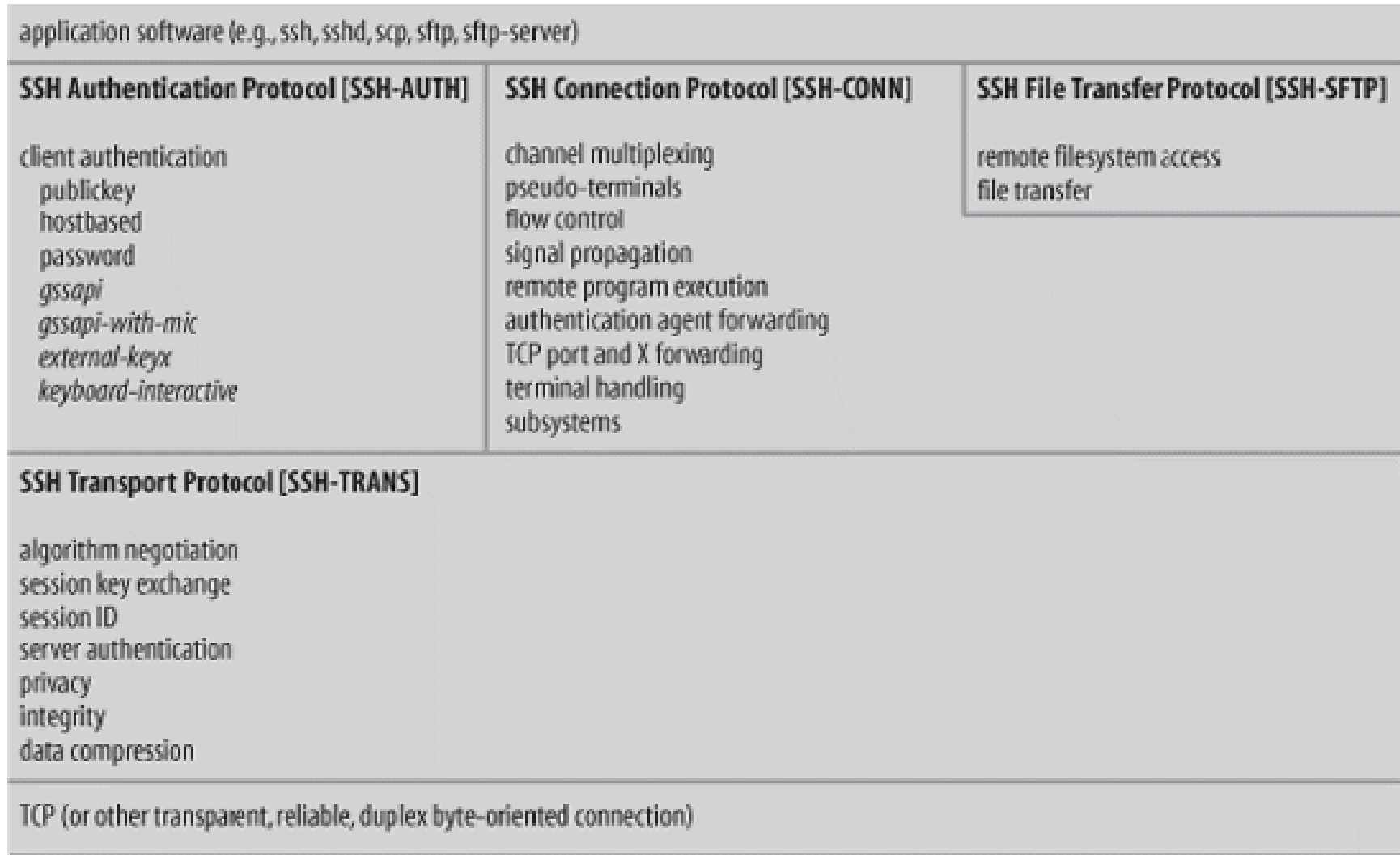
# Building Blocks

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.

101011010100010110101110101010100010010101000110101101010100010110

# Structure of the Building Blocks



10101101010100010110101110101010100010010101000110101101010100010110

# Outline

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

1010110101000101101011101010101000100101000110101101010100010110

# 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.

10101101010001011010111010101010001001010001101011010100010110



# 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).

1010110101000101101011101010101000100101000110101101010100010110

# 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.

10101101010100010110101110101010100010010101000110101101010100010110

# 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**

1010110101010001011010111010101010100010010101000110101101010100010110

# 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.

101011010100010110101110101010100010010101000110101101010100010110

# 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???

101011010100010110101110101010100010010101000110101101010100010110

# 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 3<sup>rd</sup> 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**]

101011010100010110101110101010100010010101000110101101010100010110

# Outline

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

1010110101000101101011101010101000100101000110101101010100010110



# 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

1010110101000101101011101010101000100101000110101101010100010110

# 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

1010110101000101110101110101010100010010101000110101101010100010110

# 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

101011010100010110101101010100010010101000110101101010100010110

# Outline

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

1010110101000101101011101010101000100101000110101101010100010110

# 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

10101101010100010110101110101010100010010101000110101101010100010110

# Applications of SSH

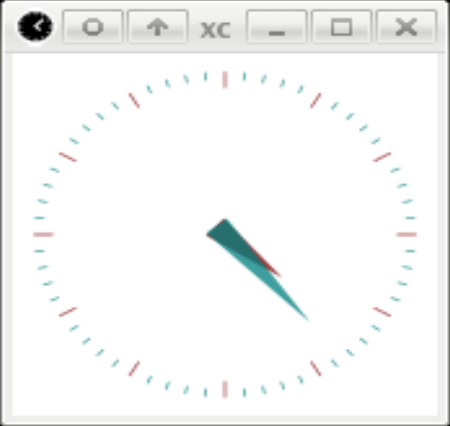
10101101010100010110101110101010100010010101000110101101010100010110

# X11 Forwarding

```

/homes/josephwu
joseph@HLT029:~ > ssh -X josephwu@hlt030
Password:
Last login: Sun Dec  3 16:21:02 2006 from hlt029.cse.ust.hk
josephwu@HLT030:~ > xclock
josephwu@HLT030:~ > xclock

```



10101101010100010110101110101010100010010101000110101101010100010110

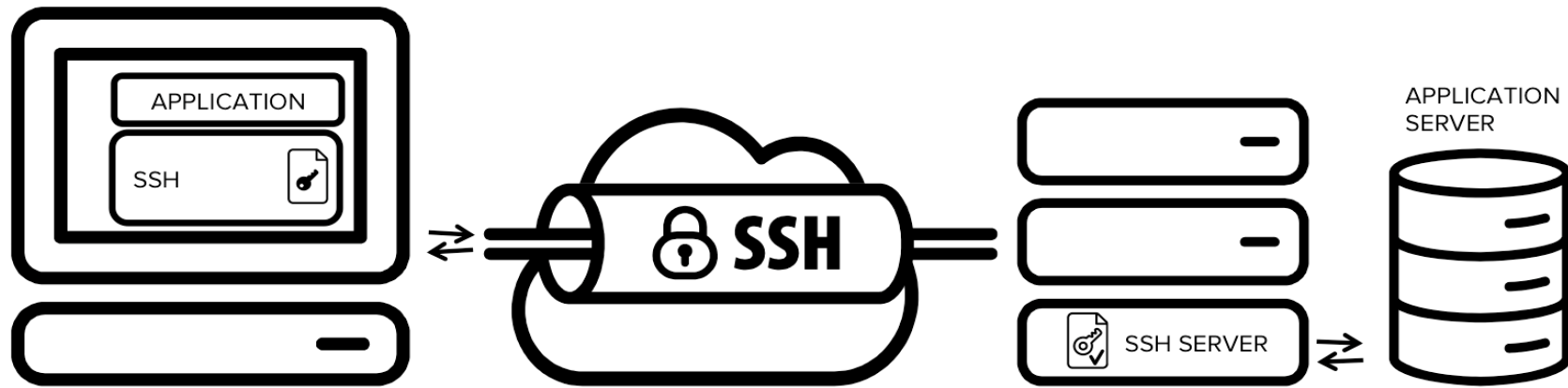
# SSH Port Forwarding

- 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.

1010110101000101101011101010101000100101010001101011010100010110



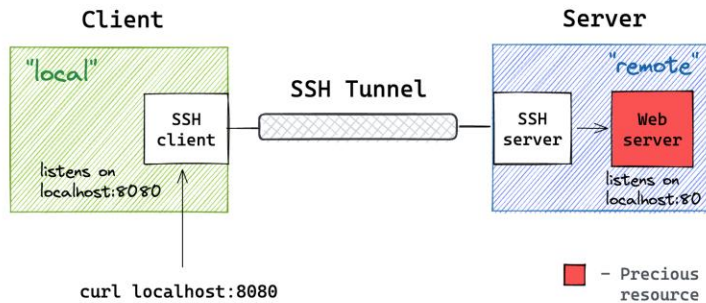
# PORT FORWARDING (CONTINUED)



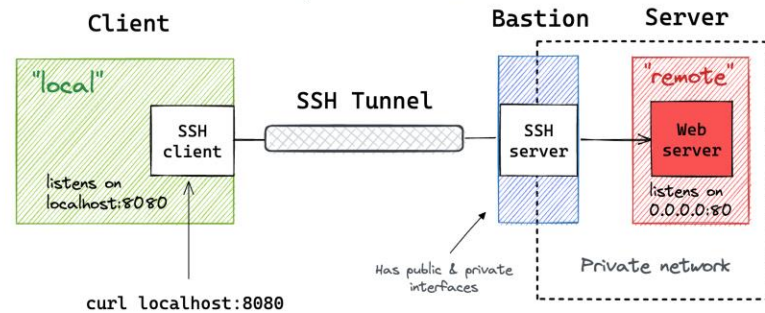
101011010100010110101110101010100010010101000110101101010100010110

# PORT FORWARDING (CONTINUED)

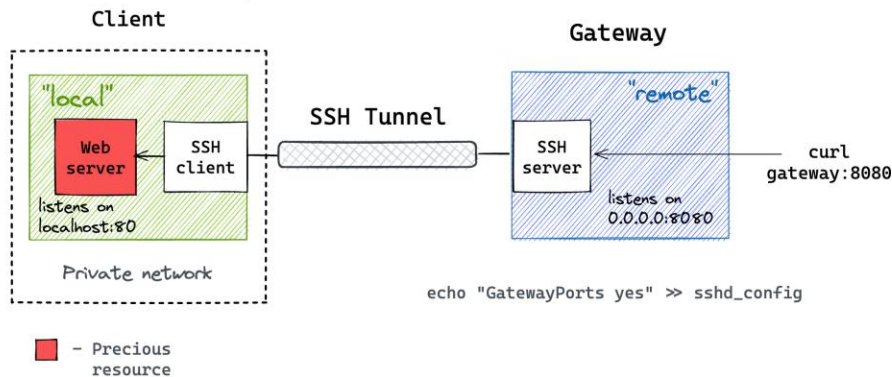
Short form → "local" address "remote" address sshd address  
`ssh -L 8080 :localhost:80 user@server`  
 Long form → `ssh -L localhost:8080:localhost:80 user@server`  
 local address tells ssh client where to start listening  
 remote address tells sshd server where to forward traffic to



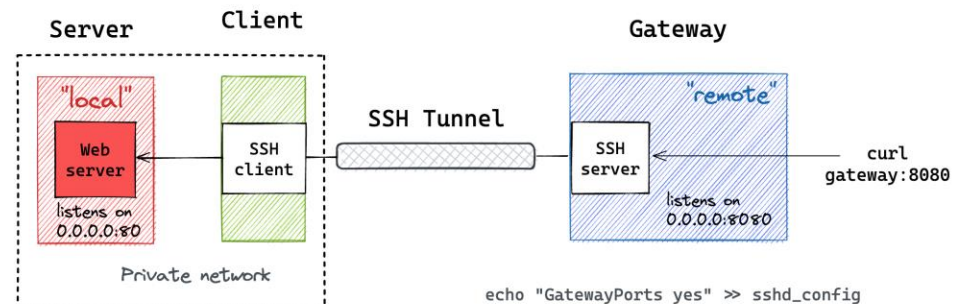
Short form → "local" address "remote" address sshd address  
`ssh -L 8080 :server:80 user@bastion`  
 Long form → `ssh -L localhost:8080:server:80 user@bastion`  
 local address tells ssh client where to start listening  
 remote address tells sshd server where to forward traffic to



"remote" address "local" address sshd address  
`ssh -R 0.0.0.0:8080:localhost:80 user@gateway`  
 remote address tells sshd server where to start listening  
 local address tells ssh client where to forward traffic to



"remote" address "local" address sshd address  
`ssh -R 0.0.0.0:8080:server:80 user@gateway`  
 remote address tells sshd server where to start listening  
 local address tells ssh client where to forward traffic to



1010110101000101101011101010101000100101000110101101010100010110

# SCP, and SFTP

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

10101101010001011010110101101011010110101000101010001001010101010001001010100010110

# 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.

10101101010100010110101110101010100010010101000110101101010100010110

# References

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

101011010100010110101110101010100010010101000110101101010100010110