SSH - Secure SHell

Lecture 24

COMP581

Slides prepared by Joseph Zhaojun Wu
Revised by Cunsheng Ding in Dec 2006

Outline

- Introduction
- Protocol details
- Applications
- References

Introduction

What is SSH?

- A set of standards and associated protocols to establish a secure channel between two computers.
- Covers authentication, encryption, and data integrity.
- Originally, a replacement of insecure applications like r-commands

Why SSH?

- Drawbacks in some *traditional* applications:
  - Authentication based on IP address
  - Authentication based on reusable password
  - Data transmitted in clear text
  - X protocol vulnerable to attack
  - Intermediate hosts can hijack sessions

Features of SSH

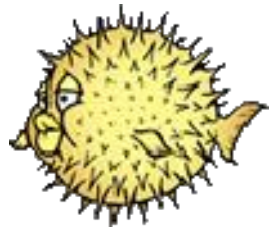
- Secure remote logins (ssh client)
- Secure remote command execution
- Secure file transfer and backup (sftp/rsync/scp)
- Public-key generation and agent for taking care of your private key
- Port forwarding and tunneling

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 SSH Communications Security (www.ssh.com) to maintain, develop and commercialize SSH, in Dec. 1995.
- Released SSH2 in 1998 based on updated SSH-2 protocol (but not compatible to SSH-1)

Brief History (cont.)

- 1999, Björn Grönvall developed OSSH based on the last open source release (1.2.12) of the original ssh program.
- "OpenBSD" then extended Grönvall's work, launched the OpenSSH project (www.openssh.org), mainly done by Markus Friedl.
- Ported to Linux, Solaris, AIX, Mac OS X, Windows (cygwin) and etc.
- Currently, OpenSSH is the single most popular SSH implementation in most of 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

SSH.com & OpenSSH

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 TSL is TCP-based and always used in WEB applications.

- There are some SSL-enhanced Telnet/FTP applications in some single hacked or patched versions. 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

3 Layers

SSH-2 Protocol has a very clean 3-layer internal architecture (RFC 4251):

- Transport Layer (RFC 4253):

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

- User Authentication Layer (RFC 4252):

  Client authentication, provide various authentication methods.

- Connection Layer (RFC 4254):

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

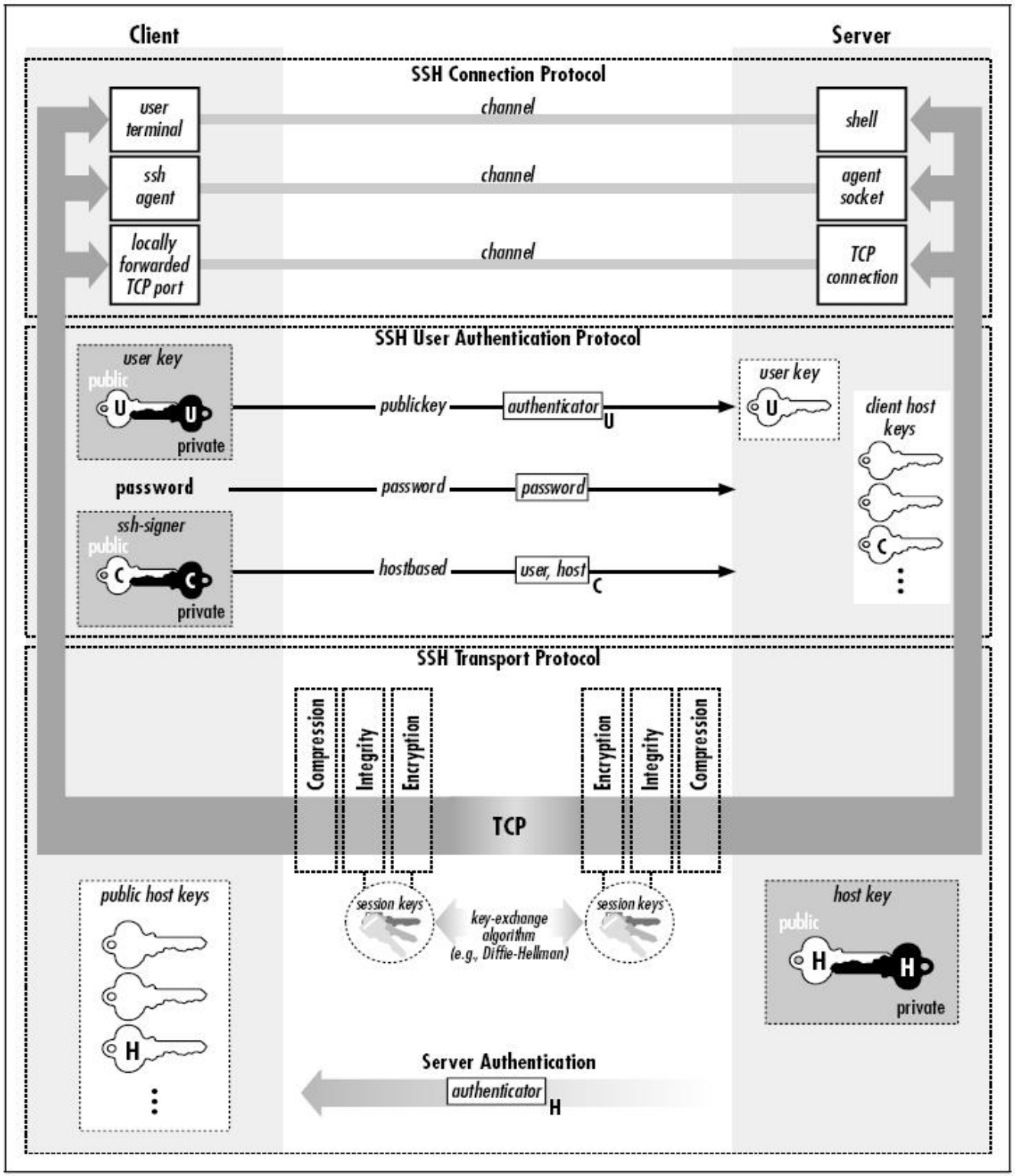
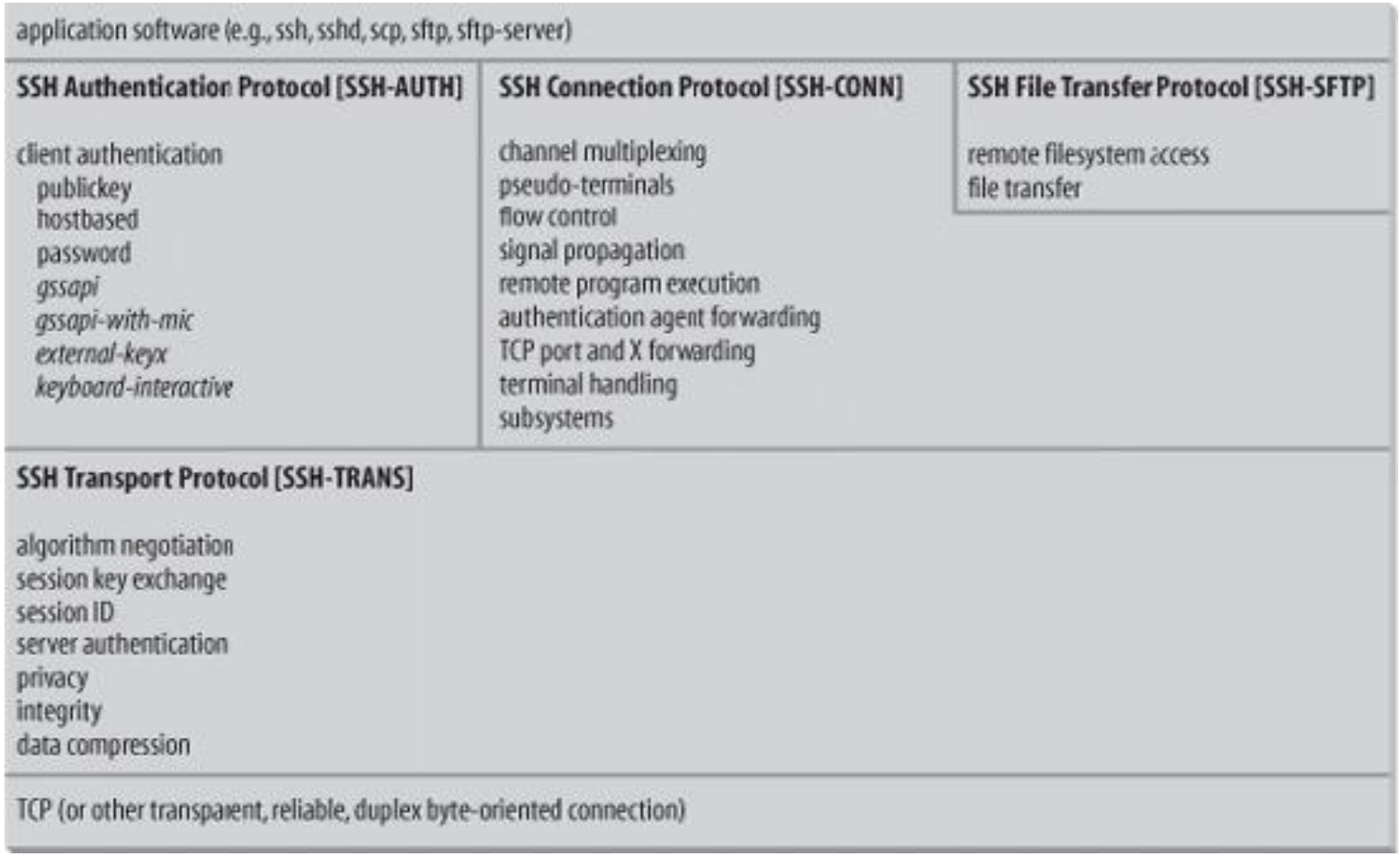


Figure 3-4. SSH-2 architecture

3 Layers


application software (e.g., ssh, sshd, scp, sftp, sftp-server)
SSH Authentication Protocol [SSH-AUTH]
client authentication
publickey
hostbased
password
gssapi
gssapi-with-mic
external-keyx
keyboard-interactive
SSH Connection Protocol [SSH-CONN]
channel multiplexing
pseudo-terminals
flow control
signal propagation
remote program execution
authentication agent forwarding
TCP port and X forwarding
terminal handling
subsystems
SSH File Transfer Protocol [SSH-SFTP]
remote filesystem access
file transfer
SSH Transport Protocol [SSH-TRANS]
algorithm negotiation
session key exchange
session ID
server authentication
privacy
integrity
data compression
TCP (or other transparent, reliable, duplex byte-oriented connection)

Outline

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

Transport Layer

- Fundamental building block of SSH.
- Providing services like initial connection, record protocol, server authentication, and basic encryption and integrity.
- After that, the client has a single, secure, full duplex stream to an authenticated server.

Connection

- Example:

```
joseph@HLT029:~ > ssh -vv joseph@freebsd
OpenSSH_4.3p2 Debian-6, OpenSSL 0.9.8c 05 Sep 2006
debug1: Reading configuration data /etc/ssh/ssh_config
debug1: Applying options for *
debug1: Connecting to freebsd [143.89.152.72] port 22.
debug1: Connection established.
```

Version Selection

- Protocol version selection:
  - Exchange a message in a form:

```
SSH-protoversion-softwareversion SP comments CR LF
```

  - Example:

```
debug1: Remote protocol version 2.0, remote software version
  OpenSSH_4.2p1 FreeBSD-20050903
debug1: match: OpenSSH_4.2p1 FreeBSD-20050903 pat OpenSSH*
debug1: Enabling compatibility mode for protocol 2.0
debug1: Local version string SSH-2.0-OpenSSH_4.3p2 Debian-6
```

  - after that, both sides switch to a nontextual, record-oriented protocol, *binary packet protocol* (the basis of SSH transport).

Parameter Negotiation: Offers from the Client

- Key exchange algorithms:

```
debug2: kex_parse_kexinit: diffie-hellman-group-exchange-
  sha1,diffie-hellman-group14-sha1,diffie-hellman-group1-sha1
```

- SSH host key types:

```
debug2: kex_parse_kexinit: ssh-rsa,ssh-dss,null
[NULL is for Kerberos authentication]
```

- Data encryption ciphers:

```
debug2: kex_parse_kexinit: aes128-cbc,3des-cbc,blowfish-
  cbc,cast128-cbc,arcfour128,arcfour256,arcfour,aes192-cbc,aes256-
  cbc,rijndael-cbc@lysator.liu.se,aes128-ctr,aes192-ctr,aes256-ctr
```

- Data integrity algorithms:

```
debug2: kex_parse_kexinit: hmac-md5,hmac-sha1,hmac-ripemd160,hmac-
  ripemd160@openssh.com,hmac-sha1-96,hmac-md5-96
```

- Data compression algorithms (optional):

```
debug2: kex_parse_kexinit: none,zlib@openssh.com,zlib
```

Parameter Negotiation: Messages back from the server:

```
debug2: kex_parse_kexinit: diffie-hellman-group-exchange-
  sha1,diffie-hellman-group14-sha1,diffie-hellman-group1-
  sha1
debug2: kex_parse_kexinit: ssh-dss
debug2: kex_parse_kexinit: aes128-cbc,3des-cbc,blowfish-
  cbc,cast128-cbc,arcfour128,arcfour256,arcfour,aes192-
  cbc,aes256-cbc,rijndael-cbc@lysator.liu.se,aes128-
  ctr,aes192-ctr,aes256-ctr
debug2: kex_parse_kexinit: hmac-md5,hmac-sha1,hmac-
  ripemd160,hmac-ripemd160@openssh.com,hmac-sha1-96,hmac-
  md5-96
debug2: kex_parse_kexinit: none,zlib@openssh.com
```

Key Exchange & Server Auth.

- After the para. negotiation, the real master key exchange is ready to go:

```
debug1: SSH2_MSG_KEX_DH_GEX_REQUEST(1024<1024<8192) sent
debug1: expecting SSH2_MSG_KEX_DH_GEX_GROUP
debug2: dh_gen_key: priv key bits set: 132/256
debug2: bits set: 513/1024
debug1: SSH2_MSG_KEX_DH_GEX_INIT sent
debug1: expecting SSH2_MSG_KEX_DH_GEX_REPLY
```

- Server authentication:

```
# server replied its public host key
debug1: Host 'freebsd' is known and matches the DSA host
  key.
debug1: Found key in /home/joseph/.ssh/known_hosts:51
debug2: bits set: 502/1024
debug1: ssh_dss_verify: signature correct
```

Derive other Keys

- Based on the shared master key, derives data encryption key and data integrity key, in both sides:

```
debug2: kex_derive_keys
debug2: set_newkeys: mode 1                [MODE_OUT   send out]
debug1: SSH2_MSG_NEWKEYS sent
debug1: expecting SSH2_MSG_NEWKEYS
debug2: set_newkeys: mode 0                [MODE_IN  receive in]
# recved the new keys from server side
debug1: SSH2_MSG_NEWKEYS received
```

- Service request: (the end of key exchange)

```
debug1: SSH2_MSG_SERVICE_REQUEST sent
debug2: service_accept: ssh-userauth
debug1: SSH2_MSG_SERVICE_ACCEPT received
```

Binary Packet Protocol

- each packet is in the following format:

```
ip_header
tcp_header
unit32    packet_length
byte      padding_length
byte[n1]  payload  n1= packet_length-padding_length - 1
byte[n2]  random padding          n2 = padding_length
byte[m]   mac                          m = mac_length
```

1. (packet_length||padding_length||payload||random padding)
   MUST be: max(a multiple of the cipher block size, 8-byte)
2. the padding is random in context and variable in length

Remarks and Question

- The Key Exchange actually produces two values:
  - a shared secret K and an exchange hash value H.
- 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)
- If compression is enabled, the data is first compressed and only then encrypted
- *How to obtain server's host key during the first log in?*

How to get host public key the 1st time?

- Two different trust models:
  - the client maintain a local database that associates each host name and corresponding public host key.
  - get the host key from a trusted 3rd party (Certification Authority)
- Another Option: host key association is NOT checked for the first login.

```
joseph@freebsd:~ > ssh hlt033
The authenticity of host 'hlt033.cse.ust.hk
  (143.89.152.142)' can't be established.
DSA key fingerprint is
  9b:1f:73:ff:d1:e1:89:91:35:97:11:20:f2:ac:f9:72.
Are you sure you want to continue connecting (yes/no)?
```

Required/Recommended Algorithm

- 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:
  - ssh-dss [Required]
  - ssh-rsa [Recommended]

Outline

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

User Authentication Layer (1)

- Runs atop of transport layer
- Relies on data privacy and integrity, provided by the transport layer
- Service ID: "ssh-userauth"
- Has access to the shared secret Session ID from transport layer
- Many authentication methods are available and they are negotiable

User Authentication Layer (2)

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

Authent. Request & Response

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

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

- Example: next page.

Example of Client Authentication

- User Authentication Example:

```
# Server side:
debug1: userauth-request for user joseph service ssh-
  connection method none
debug1: attempt 0 failures 0
Failed none for joseph from 143.89.152.138 port 60465 ssh2
# Client side
debug1: Authentications that can continue:
  publickey,keyboard-interactive
debug1: Next authentication method: publickey [failed]
debug1: Next authentication method: keyboard-interactive
Password:
#server side:
Accepted password for joseph from 143.89.152.138 port 60465
  ssh2
debug1: Entering interactive session for SSH2
```

Outline

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

Connection Layer

- Runs over the transport layer, utilizes the authentication layer
- Multiplexes the encrypted connection provided by the transport layer into several logical channels
- Channel type:
  - Interactive sessions
  - Remote command execution
  - X11: an X11 client connection
  - TCP/IP port forwarding
  - …

Connection Layer

- Channels – can be opened by either side
- To open a new channel
  - Allocate a channel number
  - Send a request to the other side, giving channel type
  - The other side either rejects or accepts and returns its channel number
  - Therefore a channel is identified by two numbers

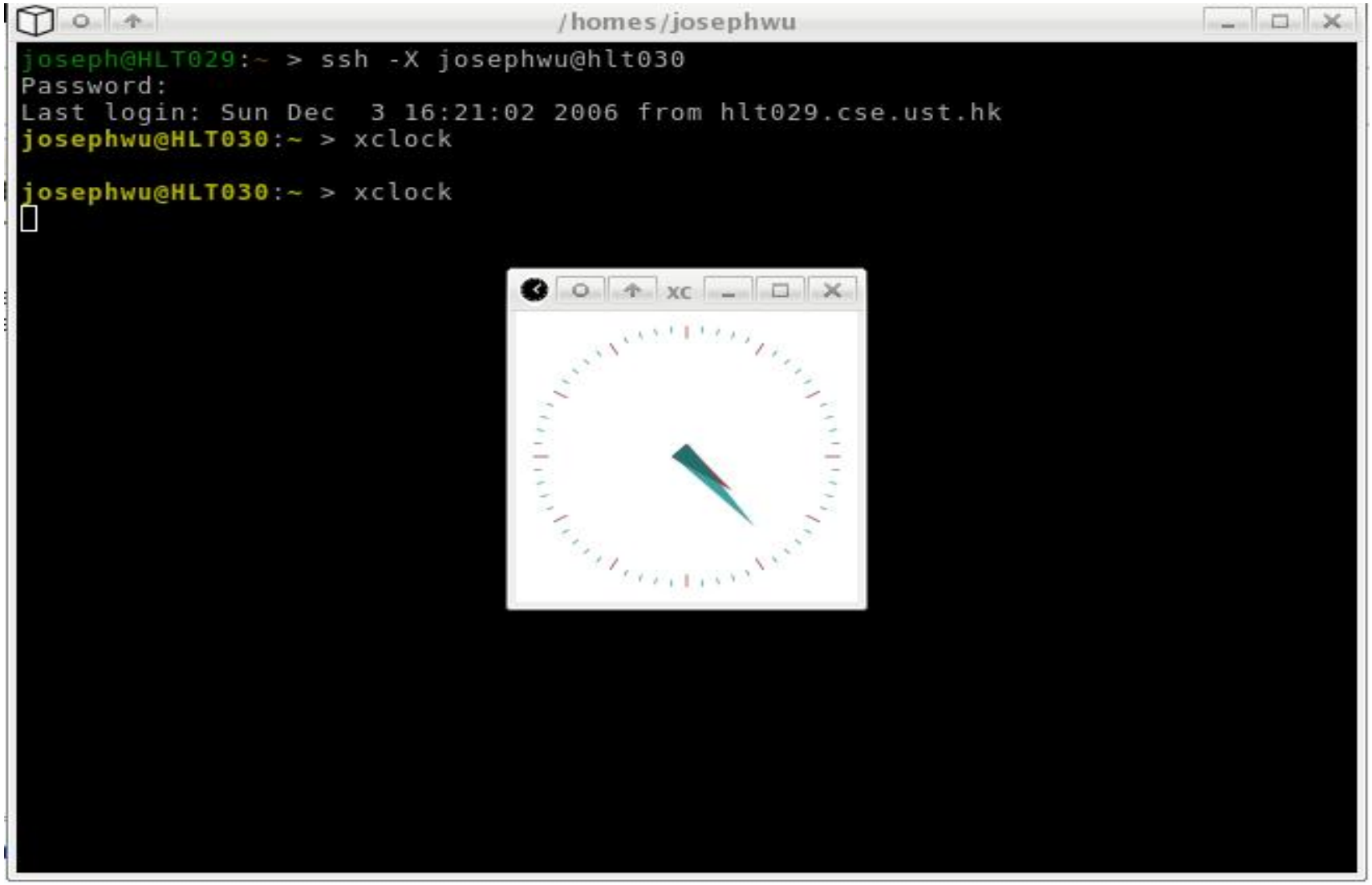
Example

- Output of the client opening a session:

```
debug1: Authentication succeeded (keyboard-interactive).
debug1: channel 0: new [client-session]
debug2: channel 0: send open
debug1: Entering interactive session.
debug2: channel 0: request pty-req confirm 0
```

Applications

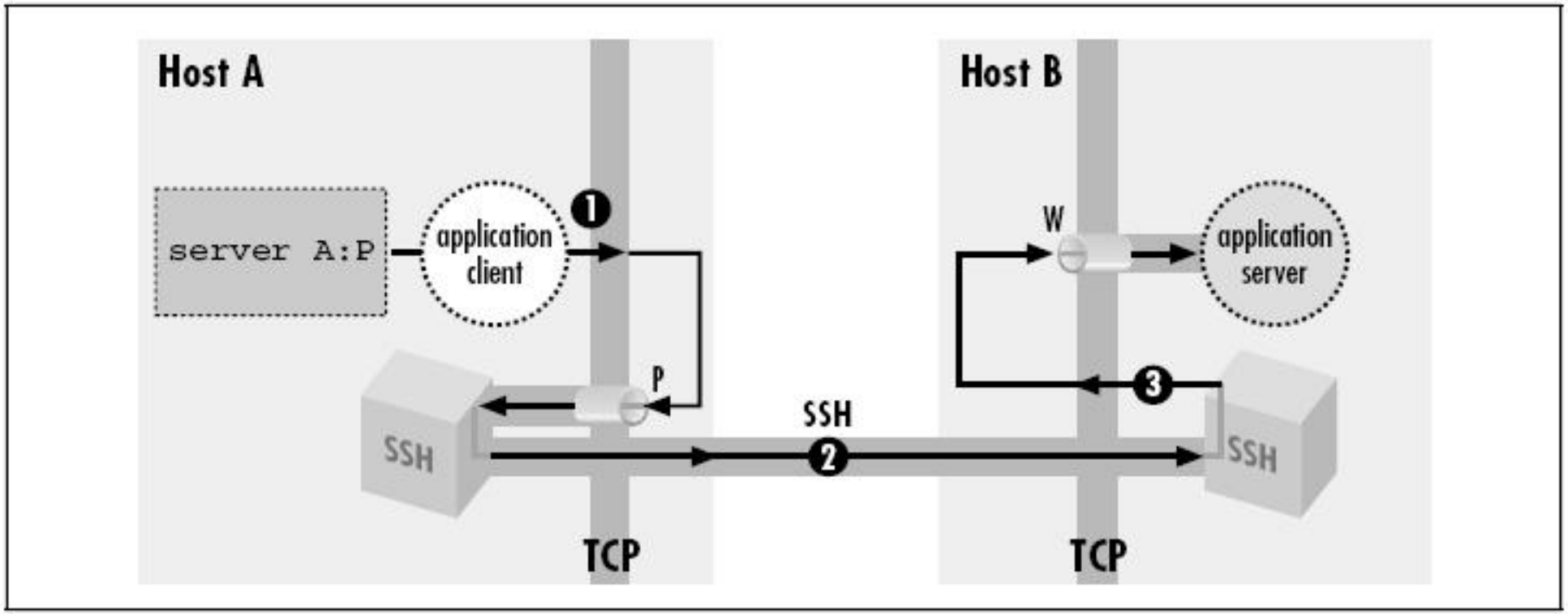
X11 Forwarding



Port Forwarding

Tunneling the connection to a remote IMAP server through SSH:

```
$ssh -L2001:localhost:143 server
```



IMAP (Internet Message Access Protocol) is an Internet standards-track protocol for accessing messages (mail, bboards, news, etc).

SCP, and SFTP

- SCP: copying files btw. hosts by using SSH for data transfer.

```
joseph@hlt029:~> scp -r mydev/* joseph@hlt030:/data/mydev
```

- SFTP: Secure FTP over SSH

```
joseph@HLT029:~ > lftp sftp://freebsd
lftp freebsd:~> user joseph
Password:
lftp joseph@freebsd:~> ls
drwxr-xr-x    5 joseph   joseph        512 Dec  3 16:20 .
drwxr-xr-x    6 root     wheel         512 Nov 17 04:18 ..
-rw-------    1 joseph   joseph         64 Dec  3 16:20 .Xauthority
-rw-------    1 joseph   joseph       4760 Dec  3 23:33 .bash_history
-rw-r--r--    1 joseph   joseph       1141 Nov 18 22:52 .bash_profile
-rw-r--r--    1 joseph   joseph       3169 Sep 19 17:42 .bashrc
```

SSH Public Key Authentication

- using "*ssh-keygen*" to generate a public/private RSA or DSA key pair, with protection from a passphrase:

```
joseph@HLT029:~ > ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/joseph/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/joseph/.ssh/id_rsa.
Your public key has been saved in /home/joseph/.ssh/id_rsa.pub.
```

- adding your public key into the server(freebsd)'s "authorized_keys" database. (~/.ssh/authorized_keys)
- connecting to the server by using public key authentication:

```
joseph@HLT029:~ > ssh freebsd
Enter passphrase for key '/home/joseph/.ssh/id_rsa':
```

SSH Agent

- Let the ssh-agent help you to do the annoying passpharse input:

```
joseph@HLT029:~ > ssh-agent bash
joseph@HLT029:~ > ssh-add
Enter passphrase for /home/joseph/.ssh/id_rsa:
Identity added: /home/joseph/.ssh/id_rsa (/home/joseph/.ssh/id_rsa)
joseph@HLT029:~ > ssh freebsd
Last login: Mon Dec  4 23:06:36 2006 from hlt029.cse.ust.
joseph@freebsd:~ > exit
joseph@HLT029:~ > ssh freebsd hostname # remote execute "hostname"
freebsd.cse.ust.hk
joseph@HLT029:~ >
```

- Usages: submit a job in remote sever, cron jobs, agent forwarding and etc.

References

- SSH: The Secure Shell The Definitive Guide 2E
- SSH FAQ
- OPENSSH Project Official Site
- SSH Communications Security

Acknowledgements

- OS used in experiments: GNU/Linux Debian *etch* and FreeBSD 6.1
- SSH-2 Client & Server: OpenSSH 4.2 & 4.3

- Some figures used in this slides are copied from the
- book "SSH, the Secure Shell – The Definitive Guide 2nd Edition" at Safari Books Online (Oreilly)