SSH: Secure Shell

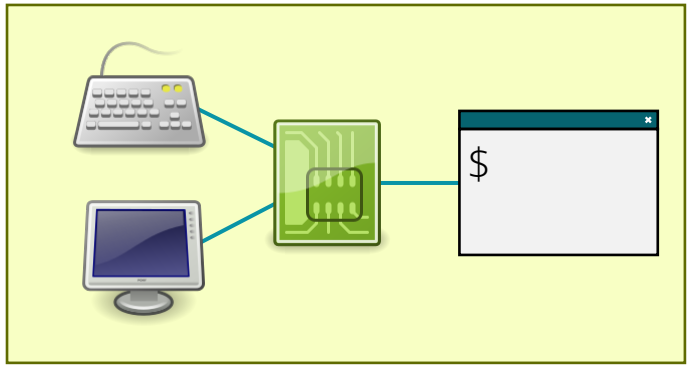
COMS10012 / COMSM0085

Software Tools

Shell

This allows you to remotely execute commands on a different machine.

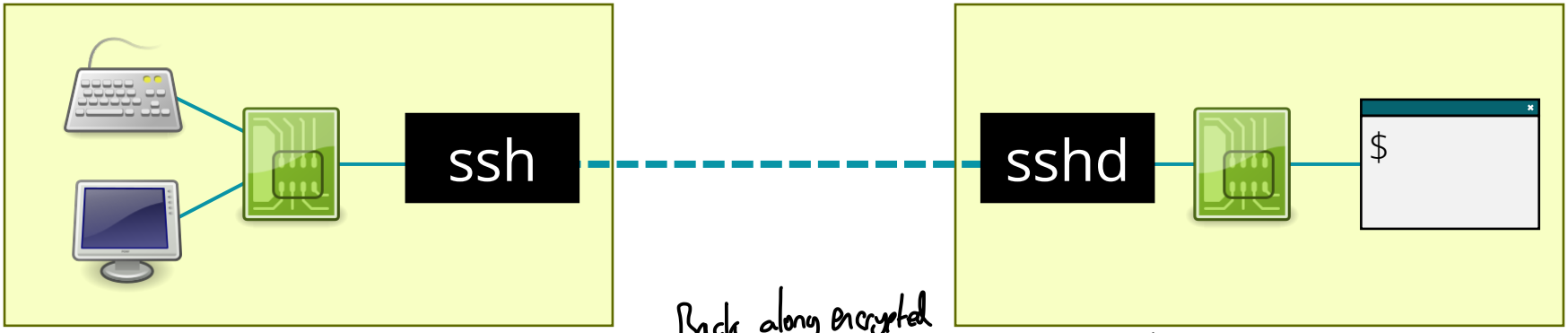
local



d = "daemon"

remote with ssh

SSH client → Encrypted Channel → daemon program



Displayed to user. ← Back along encrypted channel ← Recieves output

syntax

```
$ ssh [USERNAME@]HOSTNAME
```

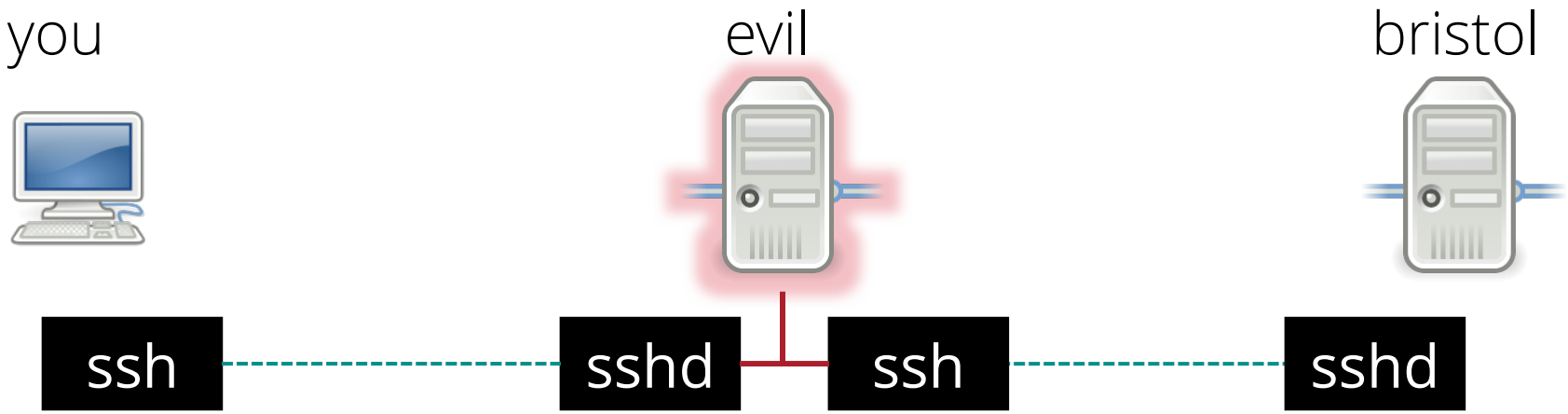
```
...
Are you sure [y/n]?
```

USERNAME is optional, if you omit it defaults to your local username

ssh ws19177@seis.bristol.ac.uk

man in the middle

pretends to be SEIS when talking to you
& pretends to be you when talking to SEIS



cryptography

SSH asks if your sure if you want to connect when you connect to a machine with a previously unseen public key or it will tell you if you already have that host key for a different

- SSH servers have a key pair for digital signatures. (public + private)
- If you connect to a new server, SSH asks you if you're sure.
- If you connect to a server and the key has changed, you get a warning.

Public key is stored in a local 'known hosts' directory

keys

This is asymmetric cryptography, public key from host decrypts their hello message
~ our public key decrypts our private message.

You can create your own key too.

- convenient: you don't have to type your password anymore.
- secure: key never leaves your machine, even if you connect to an evil server.

Details in the exercises ...

servers

`seis.bris.ac.uk`
bastion host, accessible from internet, but doesn't have your files or programs

`rd-mvb-linuxlab.bristol.ac.uk`
load balancer, connects you to a lab machine

`it#######.wks.bris.ac.uk`
lab machines, #=075637 to 075912.

SSH Notes:

- SSH is a protocal that allows you to remotely log into another computer, such as a lab machine
- When you type something, SSH encrypts it & sends it to the shell & the opposite happens for the shell's response
- Open SSH has 2 programs
  - ssh is the client which you run on your machine to connect to another machine
  - sshd is the server or daemon in UNIX-speak. It runs in the background on the machine you want to connect to.
- SSH uses port 22 by default
- To check that the SSH client is working we use the command

```
ssh localhost
```

What this does is try to initiate an SSH connection to my local machine
This command returns connection refused on my machine. This means I have SSH client (good) but no server running on my own machine, this is fine because we want to connect to a different machine anyways.

- Bristol lab machines are not accessible directly from the internet for security reasons. Instead there are 2 used to connect.

① SEIS

seis.bristol.ac.uk is reachable over SSH from the Internet & its on a university network that lets you connect further to lab machines.
You shouldn't do any work on SEIS itself, because there is very little useful software there

② Load balancer

rd-mvb-linuxlab.bristol.ac.uk. This connects you to a lab machine that is currently running & ensures that users will be more or less distributed evenly among the running machines.

To connect to SEIS:

```
ssh ws19177@seis.bristol.ac.uk
```

- The first time you do this you'll have to manually confirm this is what you want to do
- Once you say yes, this address is permenantly added to the list of known hosts

Now we can SSH into the load balancer & ∴ the lab machines.

```
ssh rd-mvb-linuxlab.bristol.ac.uk
```

I am now SSH'd into one of the UoB lab machines as ws19177 & can access everything on there.
A few more commands:

`whoami`	my username
`uname -a`	system info
`hostname`	machine name
`exit`	leaves shell

The 2 SSH commands above can be achieved in 1 jump:

```
ssh -J ws19177@seis.bristol.ac.uk ws19177@rd-mvb-linuxlab.bristol.ac.uk
```

The -J "Jumps through this host" & it can accept a CSV list of hosts if you need to connect through more than one.

- Now we can log into a lab machine remotely but we still require a password to authenticate. Let's learn how to use keys instead

Setting up SSH keys:

When you establish an SSH connection, the client on your computer & the daemon on the machine you're logging into run a cryptographic protocol to exchange public encryption keys & set up a shared secret key for the session so that both sides can encrypt & decrypt messages.

It also authenticates you to the server using one of several methods.

There are 3 main authentication factors

- Something you know (Password)
- Something you have (Encryption key)
- Something that you are (Biometrics)

An authentication method that requires 2 of these is considered 'good practice', and is called 2 factor authentication

For ssh, this means that:

- you can use a password (something you know)
- you can use a digital key (something you have)
- you can log in with a keyfile that's password protected (both)

Keys come in pairs e.g id_ed25519...

- Private keys are normally kept in a file called id_CIPHER where CIPHER is the cipher in use, this is for our eyes only
- Public keys are normally kept in a file called id_CIPHER.pub, this is the key you share with the world

This is asymetric cryptography meaning a message encrypted by one key can only be decrypted by the other.

- To generate a new key pair run the command:

```
ssh-keygen -t ed25519
```

The -t flag is to select the cryptographic algorithm

ed25519 means Edward's curve over the prime $2^{255} - 19$ cryptographic group.

- By default this will store you're new key in the .ssh folder in your home directory
- You can also give this folder a password if you want, this enhances security but is less convenient.
- To look at all your keys do

```
ls -l ~/.ssh
```

The -l flag means 'long', it provides a detailed list of everything in that directory including permissions:
All listing start with something like:

```
-rw-------
```

This is split into 4 groupings:

(-)(rw-)(---)(---)
① ② ③ ④

① This only applies to special file types e.g 'd' for directory
② This is the owner permissions, 'r' = read, 'w' = write, the 3rd char would be 'x' if executable.
③ Group permissions Group??
④ Permissions for everyone else

Also in this directory is known_hosts. This is where previously encountered public keys are stored

Set up key access on SEIS

- We need to upload our public key to the ~/.ssh directory on seis

- First we check the ssh directory exists:
  - log into SEIS with username & password
  - ls -al ~/.ssh (if it doesn't use mkdir ~/.ssh) -al is -a + -l. -a shows all hidden files
  - log out with exit

Then we'll use a command called secure copy SCP, which allows you to include remote hosts & copy files over an SSH connection.

```
SCP ~/.ssh/id_ed25519.pub "us19177@seis.bristol.ac.uk:~/.ssh"
```

notice how for the location in a non-local shell, we use double quotes, this is so the local shell doesn't expand the root symbol '~' to what it is on our local system. Instead we want the shell launched by SCP to expand it.

General syntax for SCP:

```
SCP source destination
```

where source & destination can be of the form: USERNAME@HOSTNAME:PATH. If it contains a colon, it is refering to a file on a different machine.

Now all that's left to do is append this key to the list of authorized keys in SEIS:

[SSH into SEIS]
- cd .ssh
- cat id_ed25519 >> authorized_keys This concatenates our public key to the list of authorized keys
- chmod 600 authorized_keys

This last command is less clear

Chmod is the command we use to change the owner permissions (known as mod-bits) & 600 is the pattern we want in base 8
Permissions are a bit field of 9-bits, how these bits are split is on the previous page but 600 translates to

```
rw-------
```

Meaning only I can read & write to it.

Setting up keys for lab machines:

- The initial issue with this method is that our source of authentication (our private key) is on our local machine, not on SEIS. For obvious security reasons, we don't want to upload our private key to SEIS.

- So we use an SSH feature called agent forwarding. This means that if you SSH into one machine, further SSH connections will reuse the same key

- To do this we use the -A flag

Note: logging into a machine never sends your private key, instead the machine sends you a challenge (which is usually a really long number). SSH then digitally signs this with your private key. This is your signature as the only thing that can decrypt this is your public key

The machine recieving this signature:
- Won't have the ability to create new signatures (impersonate you)
- Won't ever have access to your private key

Agent forwarding allows for challenges & signatures to be forwarded across multiple connections.

So now we'll do a similar process to transfer our public key to the lab machine.

- Log into SEIS
- Log into the lab machines
- Check if a .ssh folder exists (ls -l ~/.ssh)
  - if not make one with mkdir
- Securely copy your public key from seis/.ssh to lab/.ssh
- go to .ssh on lab machine (cd .ssh)
- cat id_ed25519.pub >> authorized_keys
- chmod 600 authorized_keys

- From now on we should be able to directly SSH to a lab machine using:

```
ssh -A -J ws19177@seis.bristol.ac.uk ws19177@rd-mvb-linuxlab.bristol.ac.uk
```

recall the -J is to 'jump' through multiple hosts

Because my private key is password protected, I'll still have to enter it twice.

Final step is to set up a config file to make the ssh command easier to type.

SSH has 2 config files it reads from:

① etc/ssh/ssh_config

This is where global settings are kept (system wide config)

② ~/.ssh/config

This is a per-use config file - the one we want to access in this case (User's config)

To see more on these config files run:

man ssh_config | less man is short for manual page (press 'q' to quit)

Anyway we want to go to:

~/.ssh/config

run touch config to make a config file

Enter the following:

```
Host seis
    HostName seis.bristol.ac.uk
    User ws19177

Host lab
    HostName rd-mvb-linuxlab.bristol.ac.uk
    ProxyJump seis
    User ws19177
```

This is all pretty self explanitory