SSH and Secure File Sharing Lab

Part 1 : SSH keys and login

- What SSH is
- What public and private keys are
- How to log in to another machine
- How Git can use SSH for authentication

Part 2: Secure file sharing

- How to send an important file safely
- How to compress and encrypt files with 7zip
- How to encrypt with `age`
- Why asymmetric encryption is useful

What is SSH?

SSH = **Secure Shell**

It lets you:

- log in to another computer safely
- run commands on that computer
- copy files safely
- use Git without typing your password every time

Real-life analogy:
SSH is like a secure door with a special key.

What is a key pair?

A key pair has two parts:

- **Public key** → safe to share
- **Private key** → keep secret

They work together.

Real-life analogy:

- The public key is like a lock you can give people.
- The private key is the only key that opens it.

Public key vs private key

Public key

- can be shared
- goes on the server
- can be used by friends to encrypt for you

Private key

- must stay with you
- should never be shared
- is used to prove it is really you

Why use keys instead of passwords?

- safer than weak passwords
- easier once set up
- works well for servers and Git
- no need to send your secret to others

Important:

If someone gets your private key, they may act as you.

Lab flow

1. Check whether SSH is installed
2. Create an SSH key pair
3. See the public and private key files
4. Add the public key to another machine
5. Log in using SSH
6. Use SSH with Git

Lab flow

7. How to compress files
8. How to encrypt files using 7zip
9. Encrypt a file with `age`
10. Decrypt a file you receive

SSH setup and login

In this section we will see:

- setup on Linux
- setup on macOS
- setup on Windows
- logging in to another machine
- Git over SSH

Linux: check SSH tools

Open a terminal and run:

```
ssh -V
ssh-keygen
```

If `ssh-keygen` by default gives a usage message, that is fine.
It usually means the tool exists.

macOS: check SSH tools

Open **Terminal** and run:

```
ssh -V
ssh-keygen
```

macOS usually already includes SSH tools.

Windows: check SSH tools

Open **cmd** or **wsl** and run:

```
ssh -V
ssh-keygen
```

Modern Windows often includes OpenSSH.

If not, install **OpenSSH Client** from Optional Features.

Create an SSH key pair

Use this command:

```
ssh-keygen -t ed25519 -C "your_email@example.com"
```

You will be asked:

- where to save the key
- whether to add a passphrase

Use the default location for now.

Why `ed25519`?

- modern
- fast
- small keys
- widely used

For beginners, it is a good default.

Where are the keys saved?

Usually inside the `.ssh` folder in your home folder.

Common files:

```
~/.ssh/id_ed25519
~/.ssh/id_ed25519.pub
```

On Windows, this is usually inside:

```
C:\Users\YourName\.ssh\
```

Which file is which?

```
id_ed25519
```

- private key
- secret
- never share

```
id_ed25519.pub
```

- public key
- safe to share

View your public key

Linux / macOS:

```
cat ~/.ssh/id_ed25519.pub
```

Windows PowerShell:

```
Get-Content $env:USERPROFILE\.ssh\id_ed25519.pub
```

You can copy this text to a server or to a friend.

Do not share your private key

Linux / macOS:

```
cat ~/.ssh/id_ed25519
```

You *can* view it, but do **not** share it.

Windows PowerShell:

```
Get-Content $env:USERPROFILE\.ssh\id_ed25519
```

Treat it like the key to your house.

Add your public key to another machine

The public key goes into this file on the remote machine:

```
~/.ssh/authorized_keys
```

That file tells the machine:

> this public key is allowed to log in

Manual way to copy the public key

1. Open your `.pub` file
2. Copy all of it as one line
3. Log in to the remote machine another way
4. Paste it into:

```
~/.ssh/authorized_keys
```

Log in to another machine

```
ssh username@remote-machine
```

Examples:

```
ssh student@192.168.1.20
ssh alice@example.com
```

If your key is accepted, you are logged in.

First-time SSH warning

You may see a message like:

> The authenticity of host ... can't be established

This means SSH is checking the identity of the remote machine.

The first time, you may need to type:

```
yes
```

What just happened?

- your computer used your **private key**
- the remote machine checked it against your **public key**
- if they match, access is allowed

You did not have to send your secret key.

Classroom exercise

- create your SSH key pair
- find the private key file
- find the public key file
- show only the public key to your partner

Classroom exercise

- add your public key to the practice server
- log in using SSH
- run:

```
whoami
hostname
pwd
```

- note what each command shows

Think about this

Why is it okay to share the public key,
but not okay to share the private key?

SSH and Git

Git can use SSH to talk to remote services like:

- GitHub
- GitLab
- Codeberg
- your own Git server

This means:

- no password every time
- your key proves your identity

SSH Git remote example

A Git remote over SSH looks like this:

```
git@github.com:user/project.git
```

or

```
ssh://git@example.com/project.git
```

Compare: HTTPS vs SSH for Git

HTTPS

- often asks for password or token
- easy to start with

SSH

- uses your SSH key
- very good for repeated use
- common for developers

Test SSH with Git hosting

Example:

```
ssh -T git@codeberg.org
```

Or for another Git server, use its SSH host.

A successful message means your key is working.

Clone with SSH

```
git clone ssh://git@codeberg.org/thejeshgn/our_class_work.git
```

Then later:

```
git push
git pull
```

without typing a password each time.

Classroom exercise

- add your public key to a Git hosting account
- test the SSH connection
- clone a repository using SSH

Optional:

- make a small change
- commit it
- push it

Part 1: summary

We learned:

- what SSH is
- what public and private keys are
- how to create a key pair
- how to log in to another machine
- how Git can use SSH for authentication

Pause

Questions?

Part 2: Sending an important document safely

In this section we will see:

- encrypting a file for someone else
- decrypting a file sent to you
- why asymmetric encryption is better

Scenario

You need to send an important document to a colleague at work.

Examples:

- contract
- salary file
- ID proof
- private report

Question: Is sending the plain file enough?

No. Email or chat alone may not be enough.

Step 1: Compress files

Why compress?

- easier to send one file
- smaller size sometimes
- cleaner when sending many files

Tools:

- **Linux:** PeaZip
- **Windows:** 7-Zip
- **macOS:** Keka

Linux: PeaZip

Use PeaZip to:

- select one or more files
- create an archive like `.zip` or `.7z`

Simple class advice:

- use `.zip` if you want broad compatibility
- use `.7z` if everyone has the right tool

Windows: 7-Zip

Use 7-Zip to:

- right-click file or folder
- add to archive
- choose `.zip` or `.7z`

7-Zip is common and easy for beginners.

macOS: Keka

Use Keka to:

- compress files into `.zip` or `.7z`
- unpack files you receive

It is simple and beginner-friendly.

Compression is not enough

A zip file is just a box.

It does **not** automatically make the content secret.

For real secrecy, we need **encryption**.

What is encryption?

Encryption changes readable data into unreadable data.

Only the right person should be able to turn it back into normal form.

Real-life analogy:
Compression is putting papers into a folder.
Encryption is locking that folder.

Demo: Encrypt files with password

- Encrypt using 7zip
- Decrypt using 7zip
- In the demo we are using peazip, a front end for 7zip

Classroom exercise

- Encrypt a file with a password
- Share the password with some folks
- See now everyone who has a password can decrypt

Why avoid this?

You *can* use this, but it has problems:

- you must share the password somehow
- that sharing step can be unsafe
- many people reuse weak passwords
- one leaked password breaks everything

Why asymmetric encryption is better

With asymmetric encryption:

- the sender uses the recipient's **public key**
- only the recipient's **private key** can decrypt
- no password needs to be shared

This is the big advantage.

Meet `age`

`age` is a simple encryption tool.

It can encrypt files:

- for an `age` public key
- or for an SSH public key

Today we will use an **SSH public key**.

Install `age` on Linux

Examples:

Ubuntu / Debian:

```
sudo apt install age
```

Install `age` on macOS

Using Homebrew:

```
brew install age
```

Install `age` on Windows

Using winget:

```
winget install age.age
```

Or install from the official release package if needed.

Check that `age` works

```
age --version
```

If you see a version number, it is installed.

Encrypt for a friend's SSH public key

Suppose your friend gave you their SSH public key file:

```
friend.pub
```

Encrypt a file like this:

```
age -R friend.pub important-document.pdf > important-document.pdf.age
```

This creates an encrypted file.

What does `-R` mean?

`-R` means **recipient file**.

That file contains the public key of the person who should be able to open the encrypted file.

Example workflow

1. Your colleague sends you their public key
2. You compress the document if needed
3. You encrypt using their public key
4. You send the `.age` file
5. Only they can decrypt it with their private key

Decrypt a file sent to you

If someone encrypted a file for you, decrypt it like this:

```
age -d -i ~/.ssh/id_ed25519 secret-file.txt.age > secret-file.txt
```

`-d` = decrypt

`-i` = identity file, usually your private key

Important rule

To encrypt for someone:

- use **their public key**

To decrypt what was sent to you:

- use **your private key**

Demo

Create a sample file:

```
echo "Salary revision draft" > important.txt
```

Encrypt it:

```
age -R friend.pub important.txt > important.txt.age
```

Demo

Decrypt it:

```
age -d -i ~/.ssh/id_ed25519 important.txt.age > important-decrypted.txt
```

Classroom exercise

- pair up with a classmate
- exchange public keys only
- each person encrypts a small text file for the other
- send the encrypted file
- decrypt what you receive

Think deeper

If two people use one shared password,
what problems can happen later?

Now compare that with using public and private keys.

Common mistakes to avoid

- sharing your private key
- encrypting with the wrong public key
- forgetting whose key is whose
- thinking compression alone is enough
- losing your private key

Two ways to protect a file

Shared password - symmetric

- simple
- must share password
- anyone with password can open

Public key encryption - asymmetric

- no shared password
- encrypt with recipient's public key
- only recipient's private key can open

Reflect

Which feels safer for important work files?

- sharing a password
- or using the recipient's public key

Why?

Part 2 summary

We learned:

- how to compress files
- why compression is not the same as encryption
- how to encrypt files with a password using 7zip (via PeaZip on Linux)
- how to install `age`
- how to encrypt using someone's SSH public key
- how to decrypt using your private key
- why asymmetric encryption avoids password sharing

Thank you!

Any questions?

Thejesh GN