Section ?: More Wireshark, advanced SSH

CSE 461 Computer Networks

Wireshark

(Not that) advanced SSH

```
ssh user@server -p port
```

SSH Keys

SSH Encryption

- SSH uses symmetrical encryption
- The session key is negotiated securely under asymmetrical encryption, upon each connection
- SSH “keys” (or passwords) are used for key negotiation
- We will learn more about cryptography in lecture
  - Take CSE 484 (Security) and CSE 490C (Cryptography) if you are interested
- We will focus on the more practical side of SSH

Why keys over passwords?

- More secure than passwords
  - Keys have completely (?) random bits
  - Passwords are vulnerable to dictionary attacks
- Easier to manage
  - Keys are kept locally and supplied automatically when you need them
  - Remembering passwords can be a pain
  - Keys can be revoked easily

Generating an SSH key pair

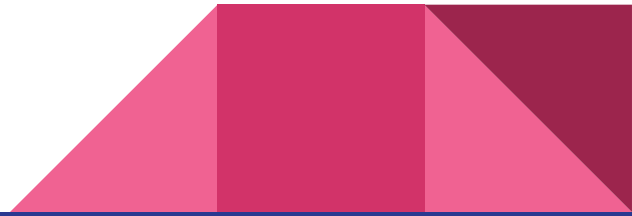
- To generate a key pair (RSA, by default): `ssh-keygen [-t type]`
  - We recommend using Ed25519 over RSA: `ssh-keygen -t ed25519`
  - Ed25519 is faster and more secure, but a lot of people are still using RSA
  - You probably have these already if you have used the CSE Gitlab
- By default, generates keys under ~/.ssh/
  - Public key: id_{rsa|ed25519|...}.pub
  - Private key: id_{rsa|ed25519|...}
  - Keep your private keys private
- Optional passphrase to protect your private keys
  - Additional passphrase-based encryption, so adversaries can't get your private keys even if your machine is compromised

Authenticating with your SSH key

- Before you can use your keys, you need to install them on the server
  - I.e. Add your public key to ~/.ssh/authorized_keys on the server
  - You can edit the file manually by logging in with your password
  - Or use `ssh-copy-id [-i path/to/private/key] someserver` (on macOS and Linux)
- Use `-i path/to/private/key` to specify a key when SSHing
  - Your id_{rsa|ed25519|dsa|...} key under ~/.ssh/ is used by default
  - Or use the `IdentityFile` option in SSH config
- When you log in, the server looks up your public key in authorized_keys and lets you in if there is a match

Server Verification (Known hosts)

- The client stores the key of every server it knows under ~/.ssh/known_hosts
- SSH stops you from connecting to a server if the server's key doesn't match the one in known_hosts
  - This often happens because someone is impersonating the server you know
  - If you trust the new server identity, simply delete its key from known_hosts



ssh-agent

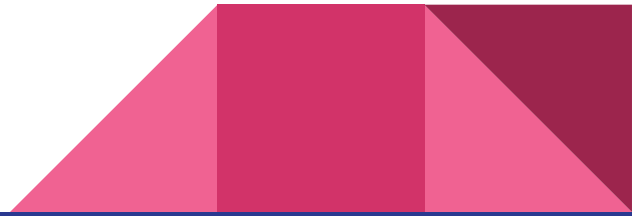
- Like a password manager for SSH keys
- Makes using passphrases easier
- `ssh-add [path/to/private/key]` to add key to ssh-agent
  - By default adds your id_{rsa|ed25519|dsa|...}
- The passphrase is remembered for the entire session

SSH Config File

SSH Config File

- Per user config at ~/.ssh/config (create if doesn't exist)
- Allows you to define hosts aliases with configurations

```
Host attu attu? recycle bicycle tricycle
  Hostname %h.cs.washington.edu
  Port 22
  User kyleyan
  IdentityFile ~/.ssh/id_ed25519
```



Simple host configs

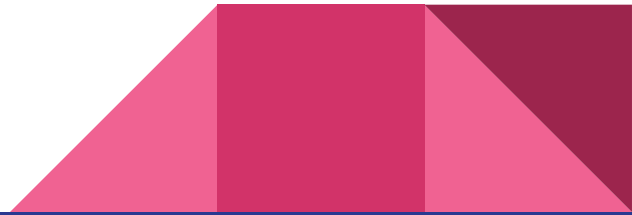
```
Host attu
  Hostname attu.cs.washington.edu
  Port 22
  User kyleyan
  IdentityFile ~/.ssh/id_ed25519
```

```
Host mininet
  Hostname localhost
  Port 2222
  User mininet
```

With the config above, I can just run `ssh attu` to connect to attu.

Equivalent to

```
ssh kyleyan@attu.cs.washington.edu -p 22 -i ~/.ssh/id_ed25519
```



A slightly more complicated config

```
Host attu attu? recycle bicycle tricycle
  Hostname %h.cs.washington.edu
  Port 22
  User kyleyan
  IdentityFile ~/.ssh/id_ed25519
```

This config defines many hosts at the same time, including a wildcard (`attu?`). Note that `%h` will be replaced by the actual value of “Host.”

With this config, I can do `ssh attu8` to connect to attu8.cs.washington.edu.

SSH Port Forwarding/Tunneling

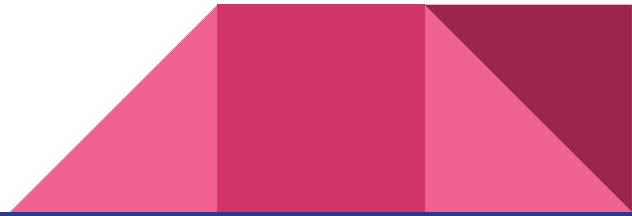
Local Forwarding (`-L`)

- Opens a local port that forwards to a remote port
- Syntax: `-L port:host:hostport`
- Use case
  - I some service running on the server, say Jupyter Lab, but bound to localhost only
  - `ssh -L 8888:localhost:8888 server`

- VSCode's Remote SSH extension provides this feature
  - Ctrl+Shift+P and search for "Forward a Port"

Remote Forwarding (`-R`)

- Opens a port on remote that forwards to a local port
- Syntax: `-R port:host:hostport`
- Requires “`GatewayPorts yes`” to be enabled on SSH server
- Use case
  - I use remote forwarding to SSH to my desktop from anywhere
  - From my desktop: `ssh -R 2222:localhost:22 publicserver.com`



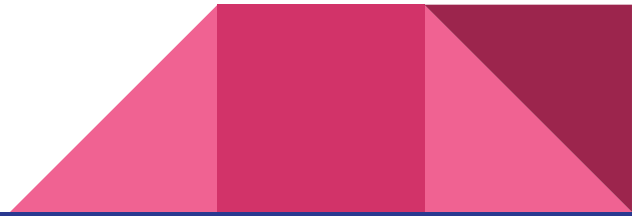
Dynamic Forwarding (`-D`)

- Uses SSH as a SOCKS proxy
- Syntax: `-D port`
- Use case
  - Use a proxy server to visit IPv6-only websites or access internal hosts
  - `ssh -D 1080 attu`
  - You probably have done this if you took 484

SSH Jump Host Proxy

Jump Host Proxy (-J)

- Use a jump host to connect to the final destination
- Syntax: `-J jumphost`
- Use case
  - You want to connect to a host behind a LAN externally, but only have SSH access to another server in that network
  - `ssh -J attu1 attu2`



X11 Forwarding

X11 Forwarding (`-X`)

- Lets you run GUI apps over SSH
- Syntax: `-X`
- Needs "`X11Forwarding yes`" enabled on server
- You might need to install an "X server" on the client if you are on Windows or macOS
  - XQuartz for macOS (and add `XAuthLocation /usr/X11/bin/xauth` to your SSH config)
  - Xming or vcxsrv for Windows
- `ssh -X attu`

You can add these forwarding / jump proxy options in SSH config, too!

Use Host * to specify options for all hosts!

Other useful SSH tricks

- VS Code Remote SSH
  - A lot of you have been using it
  - Super useful for debugging code on remote machine
- tmux
  - Keep sessions running even if you disconnect
  - Split the terminal into smaller panels
- X11 Forwarding
  - Run GUI applications over SSH
  - `ssh -X someserver`
- See `man ssh` or `tldr ssh` to learn more about advanced SSH features!