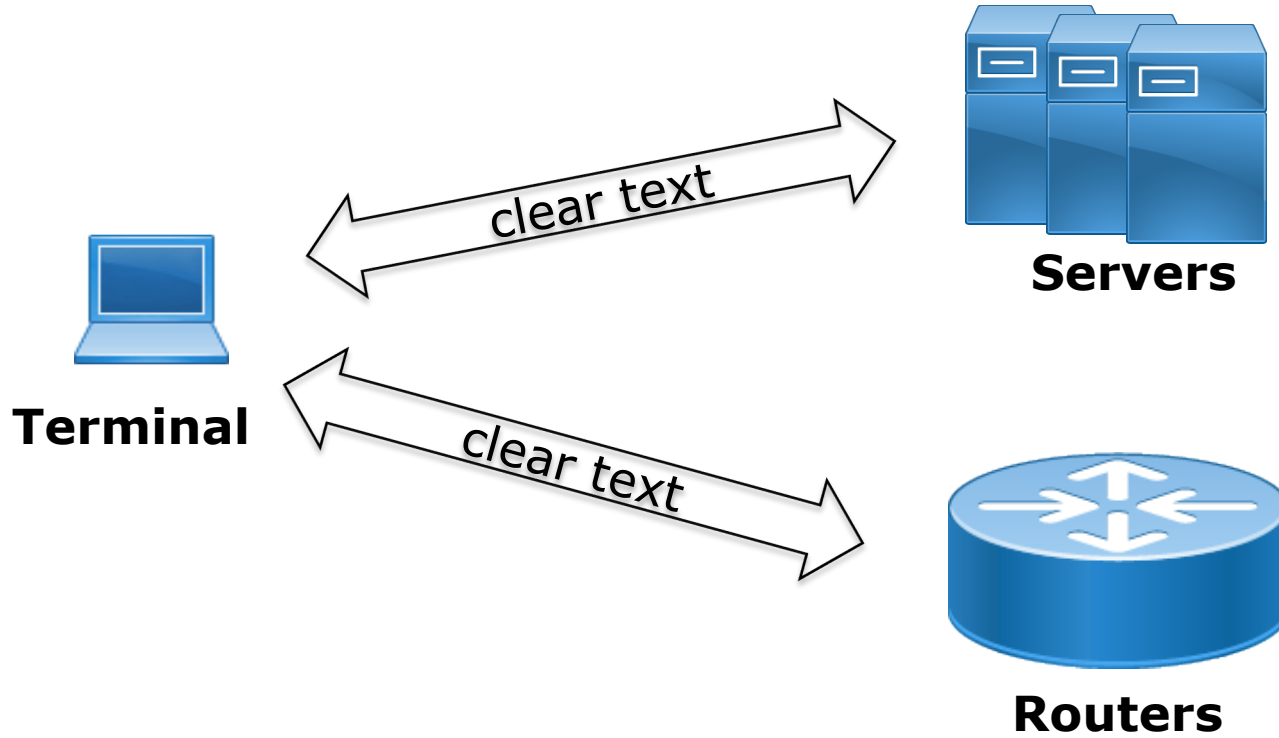


# Crypto Application 'SSH'

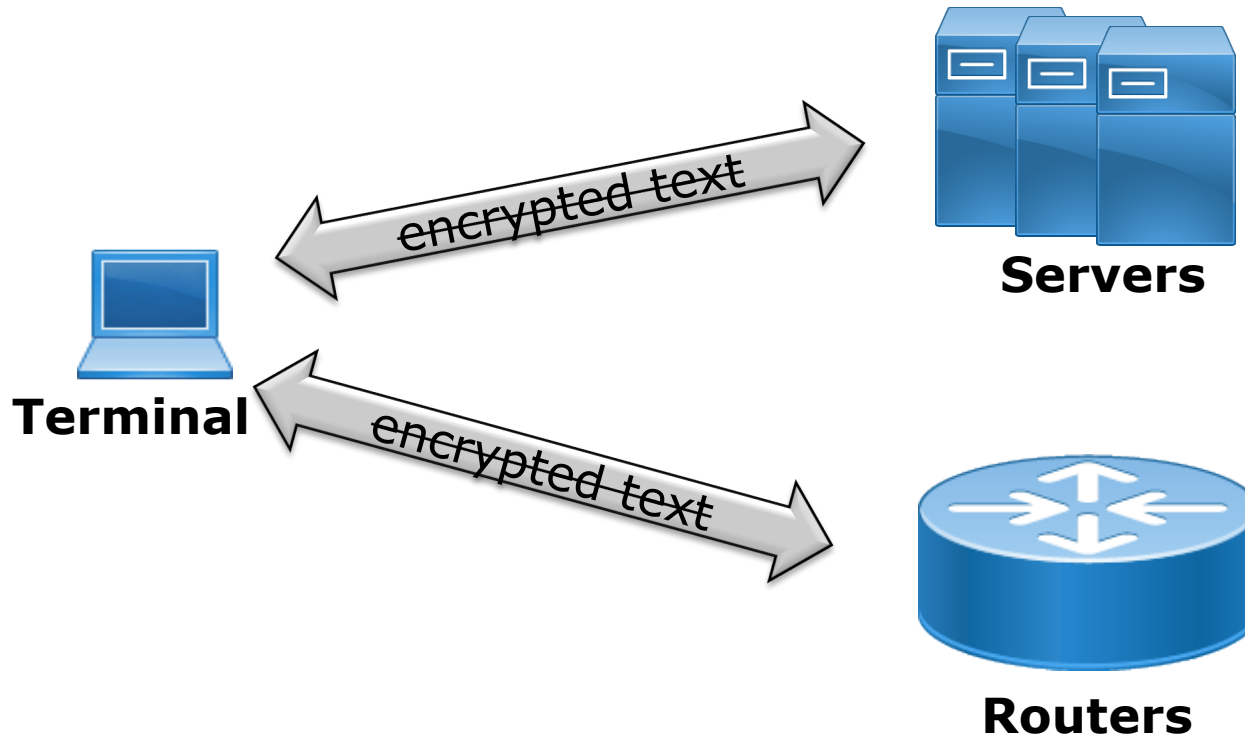
# Secure?

- **Authenticated** – I know who I am talking to
- Our communication is **Encrypted**

# Telnet



# SSH – encrypted channel



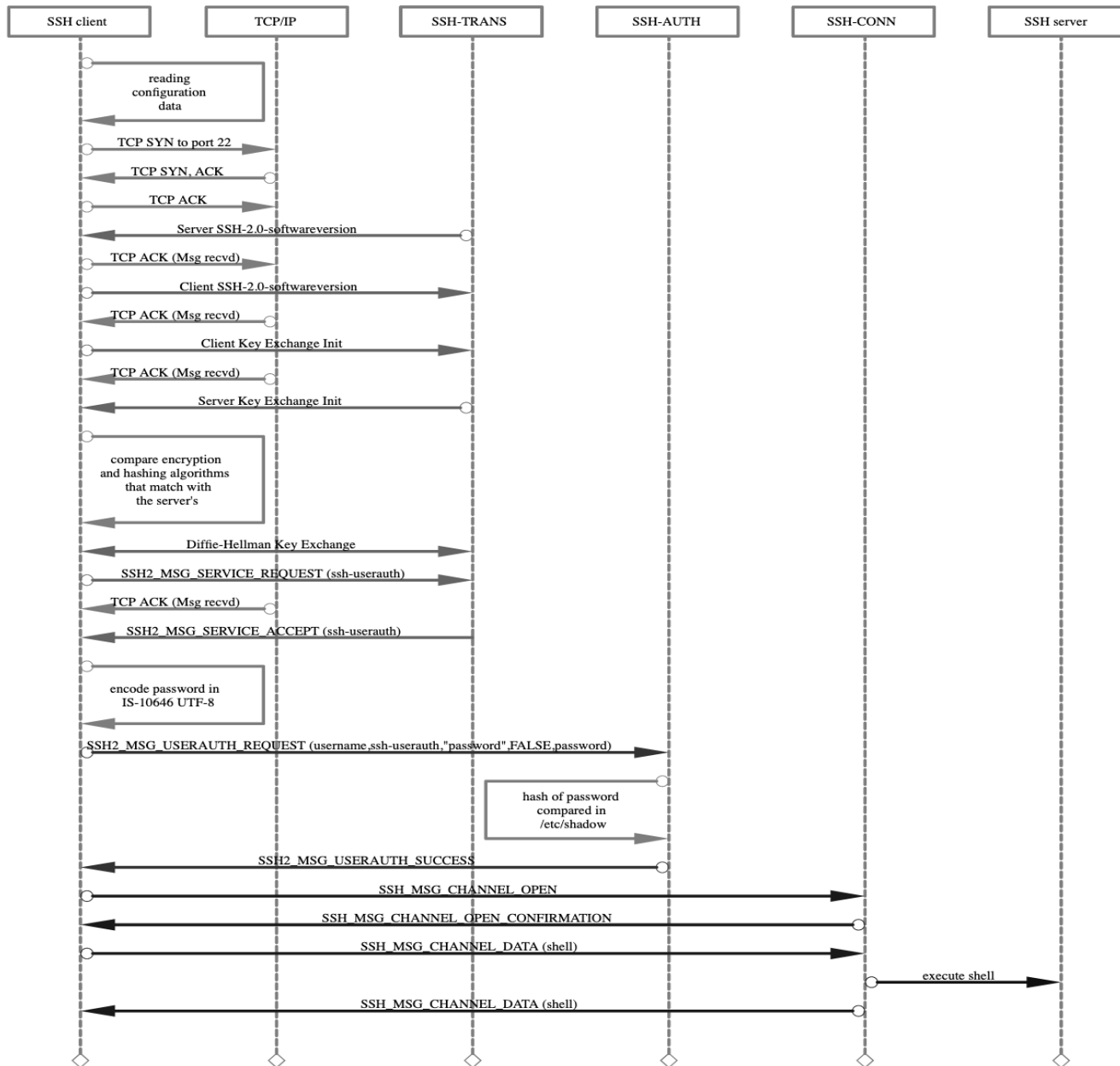
# Secure Shell (SSH)

- Authenticated and encrypted shell access to a remote host
- Client-server model
- TCP 22
- It is much more than a secure shell
  - Transport protocol (eg. SCP, SFTP)
  - Connection forwarder
    - You can use it to build custom tunnels

# SSH process

- Client-server crypto handshake
- Generate a *symmetric key* to secure the transport
- Client authenticates to server securely
- Secure communication can begin

# Under the hood



# SSH Authentication

- Client sends its username to server over the secure channel
  - Encrypted using the “**shared master key**”
- Server decrypts & checks username in the local database
  - If username not valid, tear down the SSH session!
  - If valid, the server sends back authentication method
    - Password based, or
    - Public-key based



# SSH Auth – Password

- Client sends its password
  - encrypted using the shared master secret
- Server decrypts, and checks the password
  - If match found, access is granted (shell access)

# Password Authentication

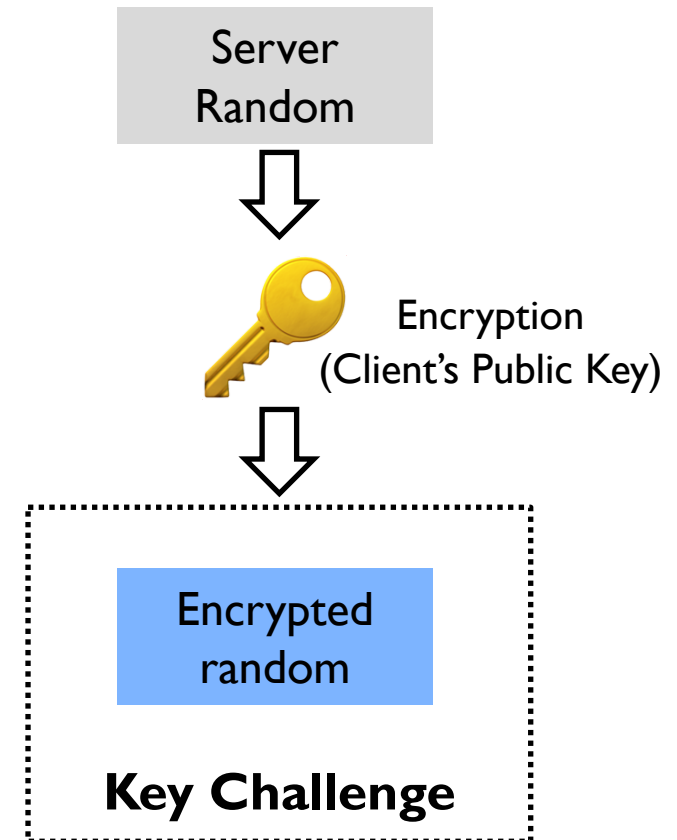
- Password Authentication is simple to set up
  - usually the default
- But allows **brute-force guessing** 😞

# SSH Authentication - Public Key

- User creates a key pair
  - public and private
    - **public key** - nonsensitive information
    - **private key** - is protected on the local machine by a strong passphrase
- Installs the public key in `$HOME/.ssh/authorized_keys` file on the target server
  - one time installation

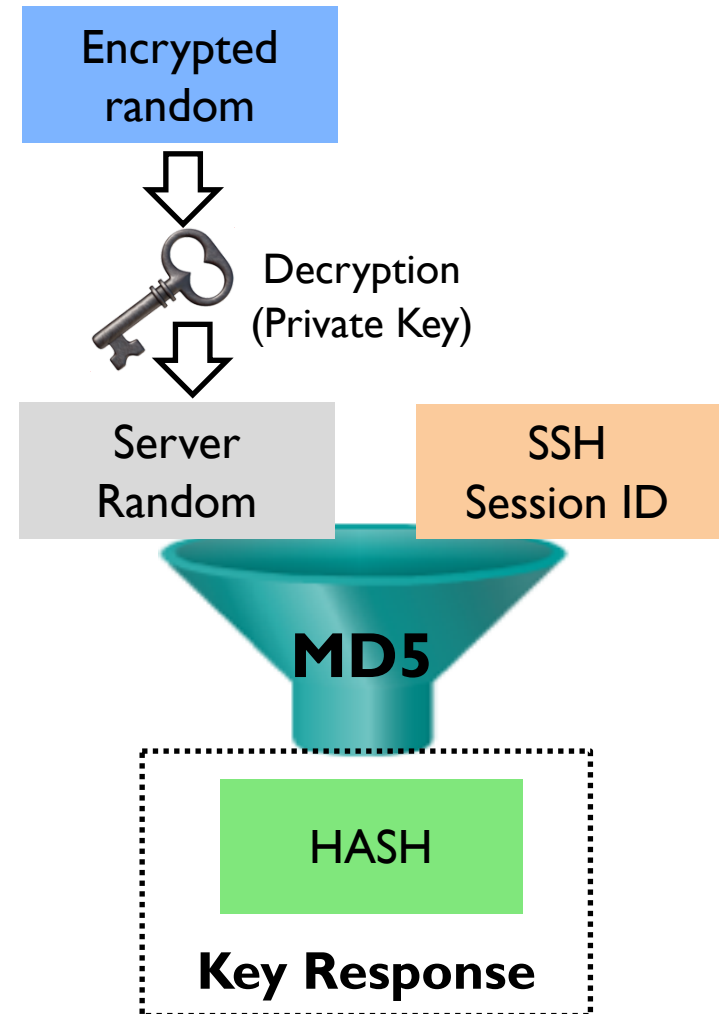
# How it works - Key Challenge

1. Client connects to server with a request to set up a key session
  - Sends KeyID for the key-pair it wants to use, and
  - Username/account-name
2. If there is a public key in the **authorized\_keys**
  - server generates a random number
  - encrypts the random using client's public
  - sends the encrypted random as a key-challenge to client



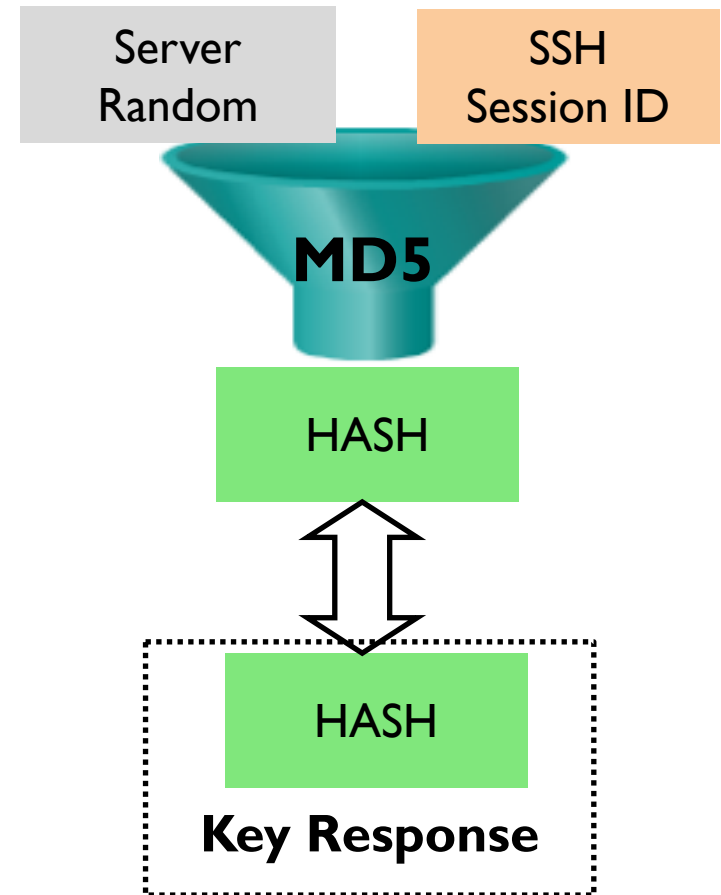
# How it works - Key Response

3. Client decrypts the random number with its private key
4. Creates an MD5 hash of the random and the session ID
  - sends back to the server as the key response
    - encrypted with shared Master key



# How it works - Access

5. The server computes its own MD5 hash and compares it with the received hash
  - random + session ID
6. If the hashes match, the user must be in possession of the private key
  - **access is granted!**



# Public Key Authentication

- Cannot derive private key from public key!
  - Cannot brute force either
- Requires one-time setup of public key on target system
- Requires unlocking private key with secret passphrase upon each connection
  - If you have setup one

# SSH Keys on Unix / MacOS

- SSH is built-in
  - UNIX
  - Linux
  - MacOS X



# Generate SSH key-pair (Unix / MacOS)

```
$/usr/home/foo> ssh-keygen -t rsa -b 4096 -C your_email@example.com
```

```
Generating public/private rsa key pair.
```

```
Enter file in which to save the key (/usr/home/foo/.ssh/id_rsa):
```

```
Enter passphrase (empty for no passphrase):
```

```
Enter same passphrase again:
```

```
Your identification has been saved in /usr/home/foo/.ssh/id_rsa.
```

```
Your public key has been saved in /usr/home/foo/.ssh/id_rsa.pub.
```

```
The key fingerprint is:
```

```
27:99:35:e4:ab:9b:d8:50:6a:8b:27:08:2f:44:d4:20 your_email@example.com
```

# SSH Keys (Unix / MacOS)

- Look inside the `.ssh` directory

`~/.ssh/id_rsa` is the private key

**DO NOT SHARE THIS FILE!**

- `~/.ssh/id_rsa.pub` is the associated public key.
- This can be shared freely without consequence.

# Saving Key on the Target Host

- You can copy the *public key* into the server's `authorized_keys` file with the `ssh-copy-id` command

```
ssh-copy-id <user@server-ip>
```

- Alternatively, you can paste in the keys using SSH:

```
cat ~/.ssh/id_rsa.pub | ssh user@serverip "mkdir -p  
~/.ssh && cat >> ~/.ssh/authorized_keys"
```

- That's it 😊

# Key generation - "Windows OS"

- Download the following tools for the lab:
  - PuTTY (the Telnet and SSH client itself)
  - PuTTYgen (an RSA and DSA key generation utility)
  - <https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html>

# Lab Exercise

1. Generate your ssh key-pair
  - Unix/Linux/OS X: *follow the examples in the slide*
  - Windows OS: *follow the lab manual [ssh\\_lab.pdf](#)*
2. Upload your SSH public key to the server
  - Windows OS: *Follow the lab manual [ssh\\_lab.pdf](#)*
  - Unix/Linux/OS X: *Follow the examples in the slide*
3. Login using your SSH public key
  - *Follow Step 4 in the lab manual [ssh\\_lab.pdf](#)*



# Questions

