SSH - Encrypted Communication - Sean Corzo 8/2023

SSH, or Secure Shell, is a network protocol

SSH is an application layer protocol, which is the 7th layer of the OSI model.

HTTPS, or Hyper Text Transfer Protocol Secure, is also another protocol that encrypts data.

HTTPS allows web browsers to communicate with servers to display websites

SSH allows for shells to enable data exchange or communication between two devices

When you use terminal or command line, you are talking to your operating system. With SSH, you can talk to remote operating systems too.

SSH breaks data down into a series of packets.

SSH operates at the Application Level.

SSH packets are created and then encapsulated as the payload at the TCP layer.

CONTENTS OF AN SSH PACKET:

- Packet Length
- Padding amount
- Payload
- Padding
- Message Authentication Code (digital signature - hash)

Payload is compressed.

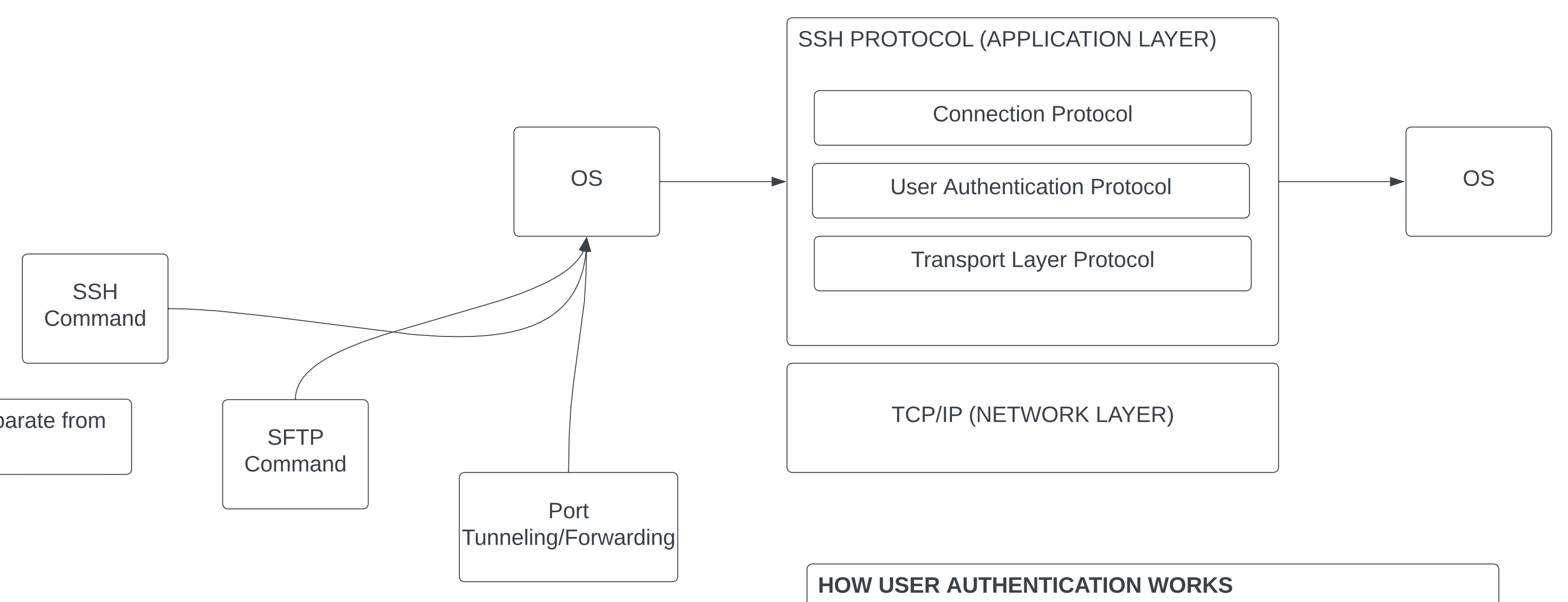
The entire packet excluding Length and Authentication Code are encrypted.

THREE COMPONENTS OF SSH PROTOCOL

1) The **TRANSPORT LAYER PROTOCOL [SSH-TRANS]** provides SERVER AUTHENTICATION, CONFIDENTIALITY AND INTEGRITY. It may optionally also provide compression. The transport layer will typically be run over a TCP/IP connection, but might also be used on top of any other reliable data stream.

2) The **User Authentication Protocol [SSH-USERAUTH]** authenticates the client-side user to the server. It runs over the transport layer protocol.

3) The **Connection Protocol [SSH-CONNECT]** multiplexes the encrypted tunnel into several logical channels. It runs over the user authentication protocol.



The SSH Command is separate from the protocol

HOW TLP WORKS

TLP effectively exchanges and secures data by:

1) AUTHENTICATING THE SERVER

An SSH security process begins when the client attempts to connect to the SSH server. SSH protocol uses public key cryptography to authenticate the server, meaning the server sends its public key to the client for confirmation. The client is able to authenticate the server by comparing this host key against a local database (aka ~/.ssh/known_hosts file) or by receiving the verification of a Certified Authority (CA).

2) NEGOTIATING THE ENCRYPTION

After the server has been authenticated, the encryption algorithm negotiation begins. Both the client and the server share their supported encryption protocols, ordered by preferability. The entry with the highest server preference that is shared across both client and server wins, making it the algorithm used to encrypt the session.

3) DETERMINING THE MASTER KEY TO DECRYPTION

Finally, once the two parties have agreed upon the encryption algorithm, they can determine the master key. SSH-2 uses the Diffie–Hellman key exchange method, in which the client and server apply the encryption algorithm to create a shared secret and perform symmetrical encryption.

Diffie–Hellman: Although the details of a Diffie–Hellman key exchange are quite complex in action, here's a basic overview of how symmetrical encryption operates to successfully secure data.

The client and server select a seed value—a large prime number.

They apply the encryption algorithm to the seed value.

Next, both the client and the server generate another prime number that acts as their personal private key.

The private key and the shared number go through the algorithm to produce a public key that each machine sends to the other.

Using their own private key, the public key of the other party, and the original seed value, both sides compute a shared key—the master key.

The end result is a shared master key that can be used by both client and host to encrypt and decrypt data. The process is thus symmetrical, allowing both sides to encrypt and decrypt while keeping their private key a secret.

After the key exchange, the client sends a service request to initiate either user authentication or connection protocol.

HOW USER AUTHENTICATION WORKS

User authorization can take several forms. The client sends a user authentication service request that specifies the method of authentication—**asymmetrical key authentication, passwords, or host-based verification**.

Many SSH users depend on the **password method**, which is completely acceptable in some environments. However, passwords are considered slightly less secure because although they are encrypted when transmitted, bots can use brute-force attacks or default passwords to gain access to a secure session.

An **asymmetrical key authentication** method relies on SSH key pairs, which do not require a password and thus avoid the possibility of exploitation.

In a **host-based system**, the host has authenticated its clients and essentially vouches for them. The server then must only authenticate the host's identity, simplifying user authentication for a host with multiple clients.

Note: RSA is the default key type when generated using the ssh-keygen command.

KEY PAIRS, IS_RSA, AUTHORIZED_KEYS

GENERATING SSH KEY PAIRS & COPYING TO REMOTE HOST

Steps for generating a key, what and where the files are:

Generate SSH keys on your local machine using the command ssh-keygen. This will create a private and public key pair. The default private key naming is ~/.ssh/id_rsa - note that a public key will be generated as well: ~/.ssh/id_rsa.pub - In order to add new keys, you will need to give them new names. For example, to create a key named my_key:

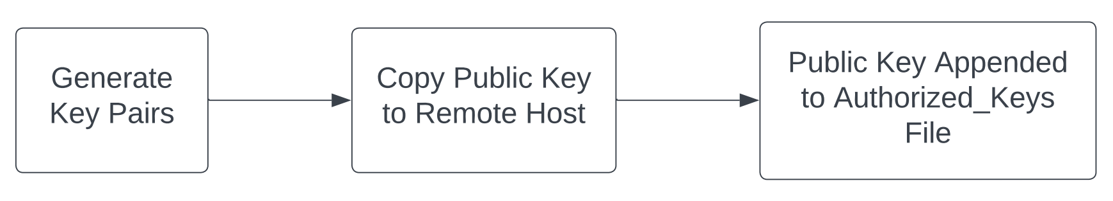
```
ssh-keygen -f ~/.ssh/my_key

# creates private key
~/.ssh/id_rsa

# and public key
~/.ssh/id_rsa.pub
```

Copy the public key to the remote host using the command ssh-copy-id user@remote_host. **This will add the public key to the remote host's authorized keys file**. The location of the authorized keys file on a remote host depends on the operating system of the remote host. On a Linux or Unix-based system, the authorized keys file is typically located in the ~/.ssh directory of the user's home directory. The file is named authorized_keys.

```
# copy public key to remote host
ssh-copy-id user@remote_host
```

# note that It prompts you for the password of the remote user (`user`) on the remote host (`host.com`) to log in initially.

```
# added to remote host here - creates .ssh folder if needed and sets correct permissions
~/.ssh/authorized_keys
```


Generate Key Pairs
Copy Public Key to Remote Host
Public Key Appended to Authorized_Keys File

MULTIPLE KEYS

MANAGING MULTIPLE PRIVATE KEYS (OPTIONAL)

In order to ssh to a host without having to specify the private key location every time on the command line, you can set up a config file locally as follows:

```
# config file
~/.ssh/config
```

When working with Git and SSH, you can set up multiple keys using the `~/.ssh/config` file. Here's an example configuration:

```
Host github-repo1
HostName github.com
User git
IdentityFile ~/.ssh/repo1_private_key

Host github-repo2
HostName github.com
User git
IdentityFile ~/.ssh/repo2_private_key
```

KNOWN_HOSTS

KNOWN HOSTS FILE (AKA SERVER AUTHENTICATION)

The known hosts file on the client machine contains a list of host keys for servers that the client has previously connected to. These files are used to ensure that the client is connecting to the correct server and that the server is the one it claims to be.

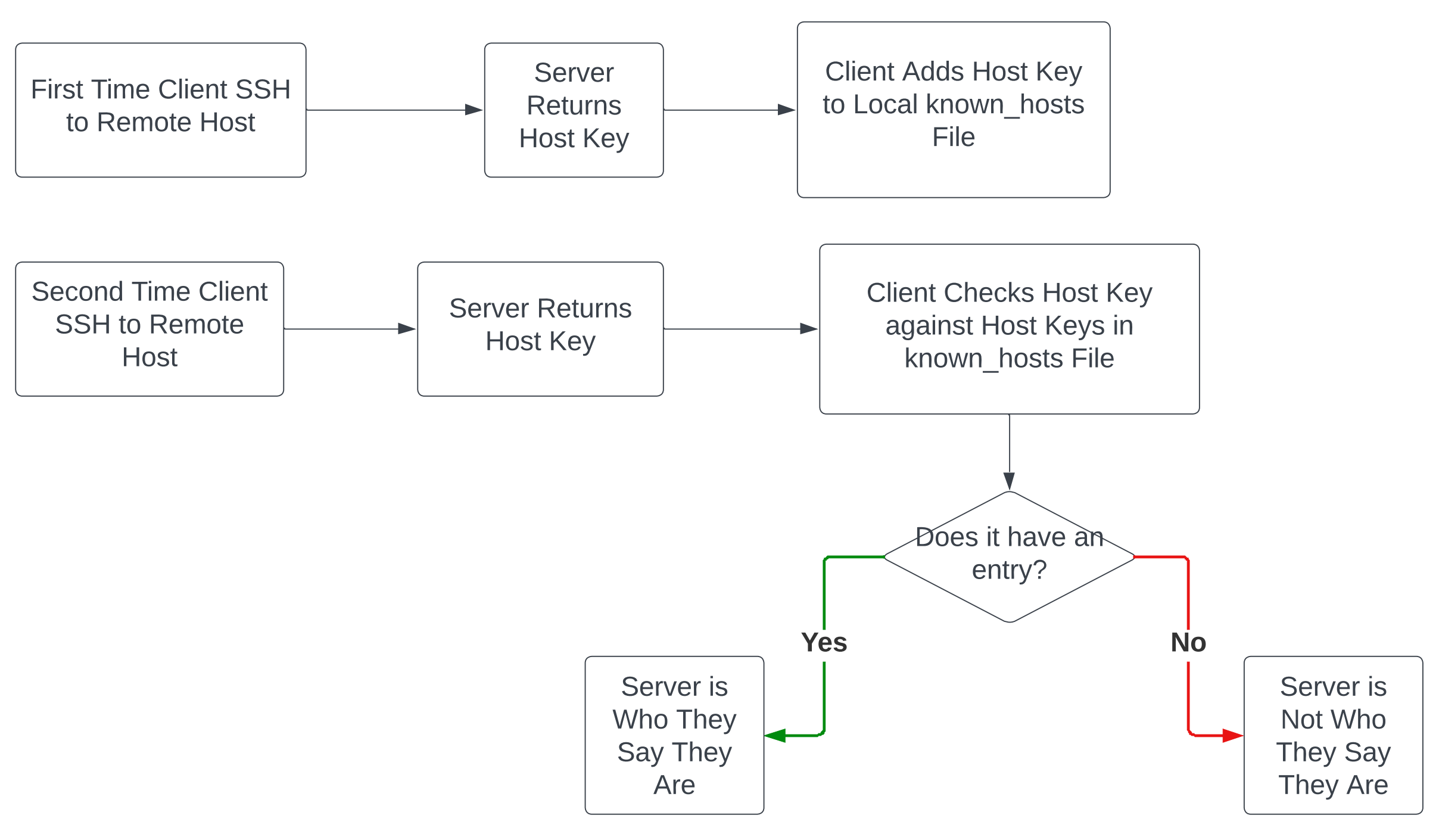
```
# on client machine
~/.ssh/known_hosts
```

The known_hosts file and the key pair authentication both serve different purposes in the SSH connection process.

The key pair authentication is used to authenticate the client to the server, and to ensure that the client is communicating with the correct server. The private key is kept on the client machine and the public key is kept on the server. When the client connects to the server, the server uses the public key to verify the identity of the client by encrypting a message using the public key and then the client decrypts it using the private key.

The known_hosts file, on the other hand, is used to authenticate the server to the client, and to ensure that the server is indeed the one it claims to be. When the client connects to a server for the first time, the server sends its host key to the client. The client then checks the host key against the known_hosts file to see if it has previously connected to the server and has a valid host key stored. If the host key is not found in the known_hosts file, the client prompts the user to confirm if they want to add the new key, this way it prevents man-in-the-middle (MITM) attacks.

While key pair authentication is used to authenticate the client to the server, the known_hosts file is used to authenticate the server to the client. Together, these two mechanisms ensure that the client is communicating with the correct server, and that the server is the one it claims to be.


First Time Client SSH to Remote Host
Server Returns Host Key
Client Adds Host Key to Local known_hosts File
Second Time Client SSH to Remote Host
Server Returns Host Key
Client Checks Host Key against Host Keys in known_hosts File
Does it have an entry?
Yes
No
Server is Who They Say They Are
Server is Not Who They Say They Are

SSH AGENT

ADD PRIVATE KEY TO SSH AGENT AND SSH TO REMOTE HOST

The primary reason for using ssh-agent is that you only need to enter your private key passphrase once as ssh-agent stores it in memory

Start the SSH agent daemon on your local machine using the command eval "$(ssh-agent -s)". This step is required in order to run ssh without the -i flag pointing to the location of your private key every time

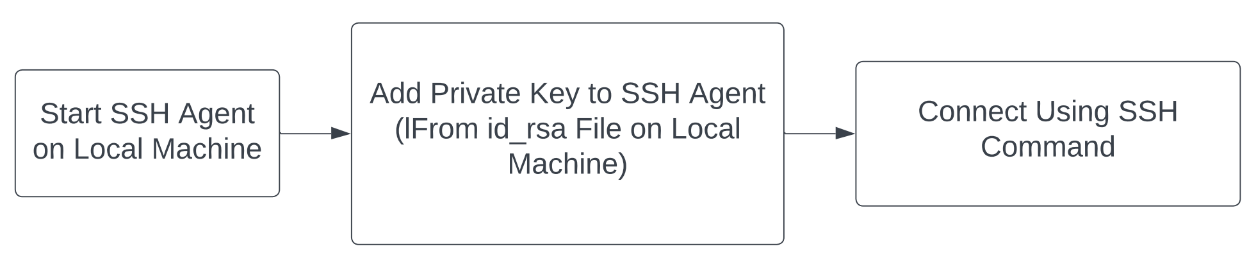
```
eval "$(ssh-agent -s)"
```

Add your private key to the SSH agent using the command ssh-add ~/.ssh/id_rsa. This step is required in order to run ssh without the -i flag pointing to the location of your private key every time

```
ssh-add ~/.ssh/id_rsa
```

Connect to the remote host using the command ssh user@remote_host

If you add multiple keys to SSH Agent and you're not using the `~/.ssh/config` file to specify key usage for different hosts, SSH Agent won't automatically determine which key to use for different hosts. Instead, SSH Agent will attempt to use the available keys in the order they were added to the agent.


Start SSH Agent on Local Machine
Add Private Key to SSH Agent (lFrom id_rsa File on Local Machine)
Connect Using SSH Command

LISTING AND REMOVING SSH-AGENT PRIVATE KEYS

How do I remove a private key from ssh agent?

To remove a private key from the ssh-agent, you can use the ssh-add command with the -d option. The syntax for this command is:

```
ssh-add -d /path/to/private_key
```

Where "/path/to/private_key" is the path to the private key file you want to remove.

Example:

```
ssh-add -d ~/.ssh/id_rsa
```

You can list the ssh-agent keys by running:

```
ssh-add -l
```

If the private key you want to remove is no longer in the list, that means it has been removed successfully.

Please note that, this command will only remove the key from the ssh-agent, it will not delete the key file.

You also can use the ssh-keygen command to delete the key:

```
ssh-keygen -f ~/.ssh/id_rsa -R hostname
```

This command will remove the key from the ssh-agent and also delete the key file.