SSH (Secure Shell)

The SSH (Secure Shell) is an access credential that is used in the SSH Protocol. In other words, it is a cryptographic network protocol that is used for transferring encrypted data over the network. The port number of SSH is 22. It allow users to connect with server, without having to remember or enter password for each system. It always comes in key pairs:

Public key – Everyone can see it, no need to protect it. (for encryption function).
Private key – Stays in computer, must be protected. (for decryption function).

Key pairs can be of the following types:

User Key – If the public key and private key remain with the user.
Host Key – If public key and private key are on a remote system.
Session key – Used when a large amount of data is to be transmitted.

What is the Secure Shell Key?
Secure Shell or SSH, is a protocol that allows you to connect securely to another computer over an unsecured network. It developed in 1995. SSH was designed to replace older methods like Telnet, which transmitted data in plain text.

Imagine a system administrator working from home who needs to manage a remote server at a company data center. Without SSH, they would have to worry about their login credentials being intercepted, leaving the server vulnerable to hackers. Instead of it after using SSH, the administrator establishes a secure connection that encrypts all data sent over the internet. They can now log in with their username and a private key, allowing them to safely execute commands on the server, transfer files, and make necessary updates, all of these without the risk of spying eyes watching their actions. This secure access is essential for maintaining the integrity of sensitive information of the company. SSH (Secure Shell) is an access credential that is used in the SSH Protocol. In other words, it is a cryptographic network protocol that is used for transferring encrypted data over the network.

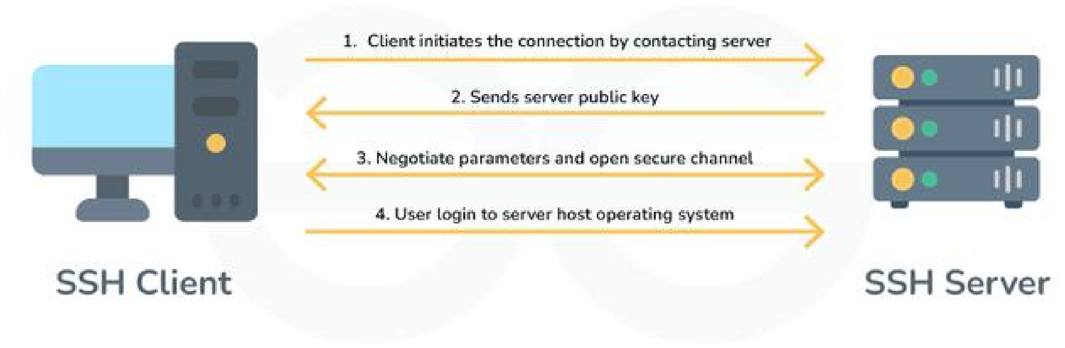
Features of SSH

Encryption: Encrypted data is exchanged between the server and client, which ensures confidentiality and prevents unauthorized attacks on the system.
Authentication: For authentication, SSH uses public and private key pairs which provide more security than traditional password authentication.
Data Integrity: SSH provides Data Integrity of the message exchanged during the communication.
Tunneling: Through SSH we can create secure tunnels for forwarding network connections over encrypted channels.

SSH Functions
There are multiple functions performed by SSH Function, here below are some functions:

- SSH provides high security as it encrypts all messages of communication between client and server.
- SSH provides confidentiality.
- SSH allows remote login, hence is a better alternative to TELNET.
- SSH provides a secure File Transfer Protocol, which means we can transfer files over the Internet securely.
- SSH supports tunneling which provides more secure connection communication.

SSH Protocol
To provide security between a client and a server the SSH protocol uses encryption. All user authentication and file transfers are encrypted to protect the network against attacks.


1. Client initiates the connection by contacting server
2. Sends server public key
3. Negotiate parameters and open secure channel
4. User login to server host operating system
SSH Client
SSH Server

SSH Authentication methods

SSH (Secure Shell) supports several authentication methods to ensure secure connections between clients and servers. Here are the primary authentication methods used in SSH:

1. Public Key Authentication

- **How it Works**: Involves a key pair consisting of a public key and a private key. The public key is stored on the server, while the private key remains with the client.
- **Process**: When the client connects to the server, the server challenges the client to prove it has the corresponding private key. If the client can sign the challenge with the private key, access is granted.
- **Advantages**: More secure than password authentication, resistant to brute-force attacks, and supports automation without passwords.

2. Password Authentication

- **How it Works**: The client provides a username and password to authenticate with the server.
- **Process**: The server verifies the credentials against the system's user database. If the username and password match, access is granted.
- **Advantages**: Simple to use but less secure than public key authentication, especially if weak passwords are used.

3. Keyboard-Interactive Authentication

- **How it Works**: Similar to password authentication but allows for additional prompts beyond just username and password.
- **Process**: The server can send a series of challenges to the client, such as asking for a password or a one-time code. This is often used in conjunction with two-factor authentication (2FA).
- **Advantages**: More flexible and can support various authentication methods, including 2FA.

4. Host-Based Authentication

- **How it Works**: Authentication is based on the client's hostname and the trust established between the client and server.
- **Process**: The server verifies the client's host credentials based on a predefined list of trusted hosts. No username/password or key is required.
- **Advantages**: Useful in trusted networks where machines can be verified without individual user authentication.

5. GSSAPI Authentication

- **How it Works**: Uses the Generic Security Services Application Program Interface (GSSAPI) for authentication.
- **Process**: This method often integrates with Kerberos for secure authentication, allowing users to authenticate without sending passwords over the network.
- **Advantages**: Enables single sign-on (SSO) capabilities and is suitable for enterprise environments.

6. Certificate-Based Authentication

- **How it Works**: Involves using X.509 certificates to authenticate users or hosts instead of traditional SSH keys.
- **Process**: The server verifies the client's certificate against a trusted certificate authority (CA). If valid, access is granted.
- **Advantages**: Offers a scalable way to manage identities and is often used in large organizations with many users and systems.

PEM key authentication

PEM key authentication is a method used in SSH (Secure Shell) to authenticate a user to a server, typically in cloud environments like AWS EC2. Here's how it works in detail:

1. Key Pair Generation

- Public/Private Key Pair: When you create a new key pair (usually in .pem format), a public key and a private key are generated.
- The public key is stored on the server (in the ~/.ssh/authorized_keys file), while the private key remains with the user and should be kept secure.

2. Adding Public Key to Server

- When launching an EC2 instance, you can specify the public key to be added to the instance's authorized_keys file automatically. This allows SSH access without needing a password.

3. SSH Authentication Process

When you try to SSH into a server using a PEM file:

- Client Sends Request: The SSH client sends a connection request to the server.

- Server Challenges Client: The server sends a challenge back to the client, asking for proof that it holds the private key corresponding to the public key in authorized_keys.
- Client Signs Challenge: The client uses its private key to sign the challenge. This involves using a cryptographic algorithm to create a unique signature based on the challenge and the private key.
- Server Verifies Signature: The server uses the public key to verify the signature. If the signature is valid, the server confirms that the client has the correct private key.
- Access Granted: If the verification succeeds, the server allows the client to log in without needing a password.

4. Security Advantages

- No Passwords: Passwords can be easily guessed or stolen, while a private key is much harder to crack due to its length and complexity.
- Public/Private Key Security: Even if someone obtains the public key, they cannot gain access without the corresponding private key.
- Multi-Factor Authentication: PEM key authentication can be combined with other forms of authentication (like password or two-factor authentication) for enhanced security.

5. Best Practices

- Keep Your Private Key Secure: Use file permissions to restrict access to your private key file. On Unix-like systems, use chmod 400 your_key.pem.
- Use Strong Passphrases: When generating keys, consider using a passphrase for your private key for additional security.
- Regularly Rotate Keys: Change your key pairs periodically to reduce the risk of compromise.