SSH and SCP - Course material

https://github.com/heig-vd-dai-course

Markdown · PDF

L. Delafontaine and H. Louis, with the help of GitHub Copilot.

Table of contents

Objectives

In this chapter, you will have a refresh about security and learn how to use the Secure Shell (SSH) protocol to connect to a remote machine. You will also learn how to transfer files from/to a remote machine with Secure Copy (SCP).

A quick reminder about security

A secure protocol means that the data sent between the client and the server is encrypted, ensuring the confidentiality and the integrity of the data.

Most secure protocols rely on cryptography to encrypt the data. The data is encrypted with an encryption key.

It is not possible to read the data without the encryption key.

Most secure protocols rely on a key pair to authenticate the client (SSH, GPG etc.). The key pair is composed of a public key and a private key.

The public key can be shared with anyone. The private key must be kept secret.

If Alice wants to send a message to Bob, Alice encrypts the message with Bob's public key. Bob decrypts the message with his private key.

If Bob wants to send a message to Alice, Bob encrypts the message with Alice's public key. Alice decrypts the message with their private key.

If Bob wants to prove that she is the author of a message, Bob encrypts the message with their private key. Everyone can decrypt the message with Bob's public key. If the message can be decrypted, it means that Bob is the author of the message.

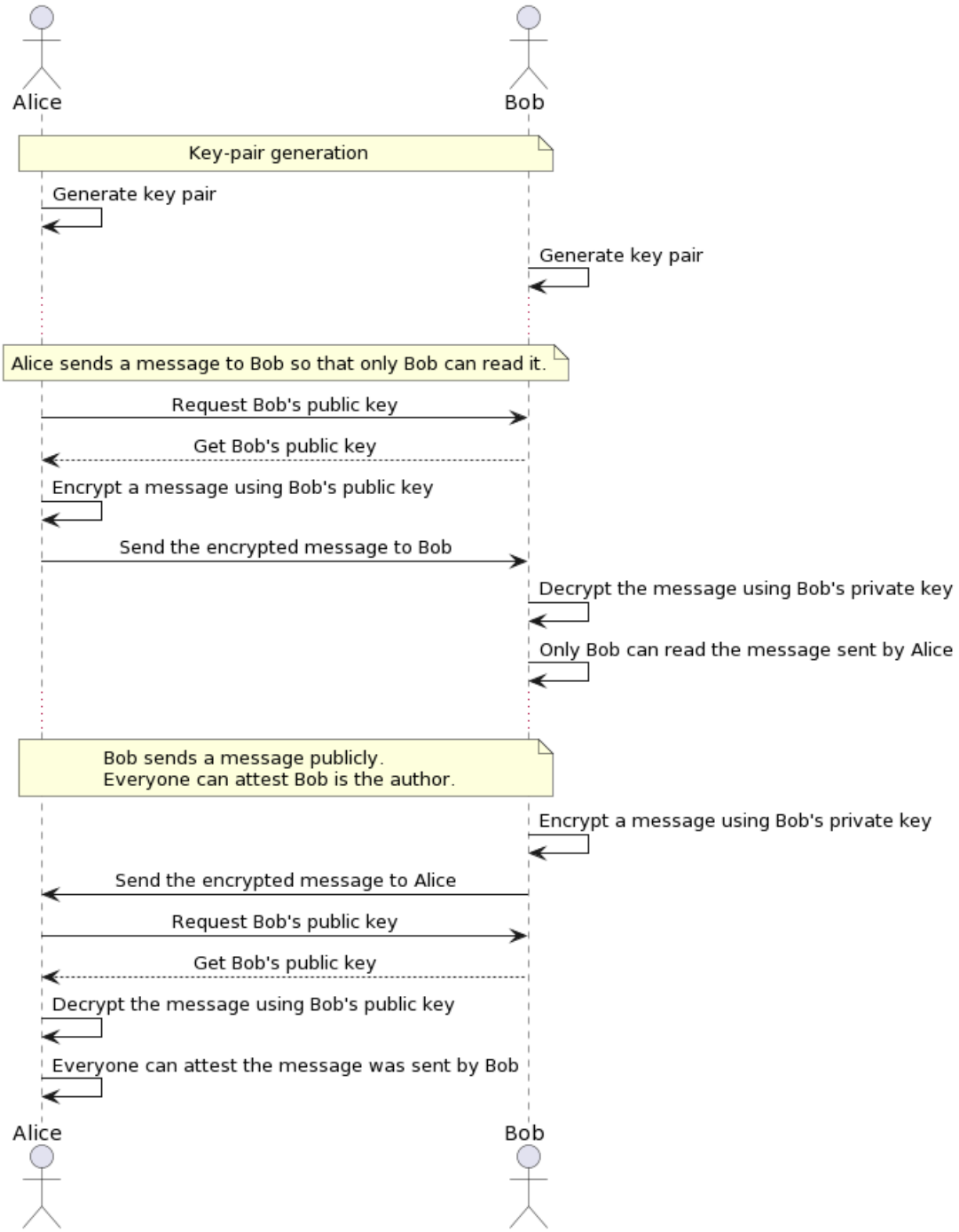
In the case of SSH and GPG, the public key is used to encrypt the data and the private key is used to decrypt the data.

This principle is used to authenticate the client: the client sends the public key to the server. The server uses the public key to encrypt a message and sends it to the client. The client uses the private key to decrypt the message.

If the client is able to decrypt the message, it means that the client has the private key. The client is authenticated.

The private key is a file that must be kept secret. If someone has access to the private key, this person can impersonate the client.

The following diagram illustrates the communication between the client and the server:



SSH

SSH is a protocol that allows you to connect to a remote machine. It is a replacement for the Telnet protocol.

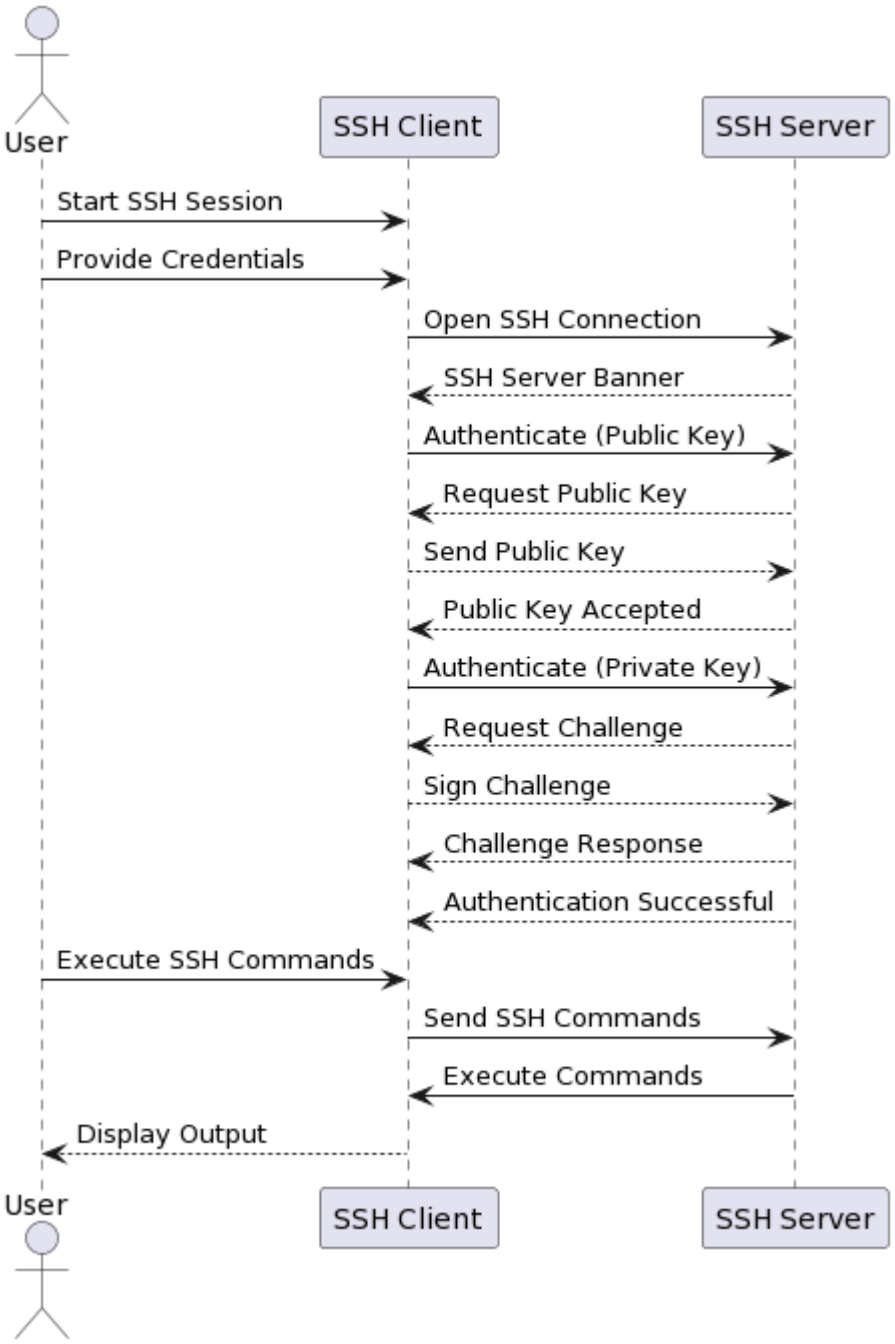
The SSH protocol is described in many RFCs:

- RFC 4250
- RFC 4251
- RFC 4252
- RFC 4253.
- RFC 4254

The SSH protocol uses the TCP protocol on port 22. It uses public/private keys to authenticate the client. It is also possible to use a password to authenticate the client but it is not recommended because it is less secure.

The SSH protocol is used to connect to a remote machine. It is also used to copy files from/to a remote machine with SCP.

SSH is also the name of the software that implements the SSH protocol. It is available on most operating systems and is considered as a standard.



SSH key algorithms

SSH supports different key algorithms. The most common are:

- RSA
- DSA
- ECDSA
- Ed25519

Out of the four algorithms, Ed25519 and ECDSA are the most recent and are considered as more secure than RSA and DSA.

SSH key fingerprint

The SSH key fingerprint is a hash of the public key. It is used to identify the public key.

The fingerprint is displayed when the key is generated. It is also displayed when the key is used to connect to a remote machine.

The fingerprint is used to verify that the public key is the same as the one used to connect to the remote machine.

This can help detect man-in-the-middle attacks.

Public keys are stored in the `~/.ssh/known_hosts` file. The file contains the fingerprint of the public key and the hostname of the remote machine.

SSH key generation

The SSH key generation is done with the `ssh-keygen` command.

The `ssh-keygen` command generates a key pair composed of a public key and a private key.

By default, the key pair is generated in the `~/.ssh` directory. The private key is stored in the `~/.ssh/<key>` file and the public key is stored in the `~/.ssh/<key>.pub` file.

The private key must be kept secret. The public key can be shared with anyone.

A passphrase can be used to protect the private key. The passphrase is used to encrypt the private key. The passphrase must be entered every time the private key is used.

Alternatives

Alternatives are here for general knowledge. No need to learn them.

- *None for now*

Missing item in the list? Feel free to open a pull request to add it!

Resources

Resources are here to help you. They are not mandatory to read.

- Introduction to SSH

Missing item in the list? Feel free to open a pull request to add it!

SCP

SCP is a protocol that allows you to copy files from/to a remote machine. It is a replacement for the FTP protocol.

The SCP protocol relies on the SSH protocol to authenticate the client.

SCP is an illustration of the client/server architecture using the SSH protocol.

Alternatives

Alternatives are here for general knowledge. No need to learn them.

- FTP
- SFTP
- Rsync

Missing item in the list? Feel free to open a pull request to add it!

Resources

Resources are here to help you. They are not mandatory to read.

- *None for now*

Missing item in the list? Feel free to open a pull request to add it!

Practical content

Install and configure the SSH client and SCP

In this section, you will install and configure SSH on your operating system.

Install the SSH client

The SSH client is available on most operating systems.

You certainly already have it installed on your operating system as you used it in the Git, GitHub and Markdown chapter.

If not, follow the instructions below to install it.

Check the installation

Open a terminal and type `ssh -V`.

The output should be similar to this:

```
OpenSSH_9.4p1, OpenSSL 3.1.3 19 Sep 2023
```

Start and configure the SSH server

Start the SSH server

Pull the latest changes from the previously cloned heig-vd-dai-course/heig-vd-dai-course-code-examples repository or clone it if you have not done it yet.

Explore the `20-ssh-and-scp` directory containing the OpenSSH server example with Docker Compose. Make sure to show hidden files and directories to see the `.env` file.

In the `20-ssh-and-scp` directory, run the following command:

```
# Start the SSH server in foreground
docker compose up
```

The output should be similar to the following:

```
[+] Running 8/8
 openssh-server 7 layers [           ]      0B/0B
Pulled                                                                  5.5s
   a43e029ba662 Pull
complete
1.2s
   07a0e16f7be1 Pull
complete
0.6s
   e3303cebae64 Pull
complete
0.5s
   35c9530ef606 Pull
complete
1.7s
   693b3eff61d8 Pull
complete
1.1s
   74b7ec7c3a46 Pull
complete
2.2s
   9bf5daa7bd38 Pull
complete
1.7s
[+] Running 2/2
 Network 20-ssh-and-scp_default
Created                                                                 0.2s
 Container 20-ssh-and-scp-openssh-server-1
Created                                                                 0.2s
Attaching to 20-ssh-and-scp-openssh-server-1
20-ssh-and-scp-openssh-server-1  | [migrations] started
20-ssh-and-scp-openssh-server-1  | [migrations] no migrations found
20-ssh-and-scp-openssh-server-1  | usermod: no changes
20-ssh-and-scp-openssh-server-1  |
20-ssh-and-scp-openssh-server-1  |
20-ssh-and-scp-openssh-server-1  |
20-ssh-and-scp-openssh-server-1  |
20-ssh-and-scp-openssh-server-1  |
```

```
20-ssh-and-scp-openssh-server-1  |
20-ssh-and-scp-openssh-server-1  |
20-ssh-and-scp-openssh-server-1  |
20-ssh-and-scp-openssh-server-1  |
20-ssh-and-scp-openssh-server-1  |    Brought to you by linuxserver.io
20-ssh-and-scp-openssh-server-1  |
20-ssh-and-scp-openssh-server-1  |
20-ssh-and-scp-openssh-server-1  | To support LSIO projects visit:
20-ssh-and-scp-openssh-server-1  | https://www.linuxserver.io/donate/
20-ssh-and-scp-openssh-server-1  |
20-ssh-and-scp-openssh-server-1  |
20-ssh-and-scp-openssh-server-1  | GID/UID
20-ssh-and-scp-openssh-server-1  |
20-ssh-and-scp-openssh-server-1  |
20-ssh-and-scp-openssh-server-1  | User UID:    911
20-ssh-and-scp-openssh-server-1  | User GID:    911
20-ssh-and-scp-openssh-server-1  |
20-ssh-and-scp-openssh-server-1  |
20-ssh-and-scp-openssh-server-1  | User name is set to <username>
20-ssh-and-scp-openssh-server-1  | sudo is disabled.
20-ssh-and-scp-openssh-server-1  | ssh-keygen: generating new host keys: RSA ECDSA
ED25519
20-ssh-and-scp-openssh-server-1  | sshd is listening on port 2222
20-ssh-and-scp-openssh-server-1  | User/password ssh access is enabled.
20-ssh-and-scp-openssh-server-1  | [custom-init] No custom files found, skipping...
20-ssh-and-scp-openssh-server-1  | [ls.io-init] done.
```

The SSH server should now be started.

Check the SSH server configuration

A Docker volume is used to store the SSH server configuration. The volume is mounted in the `./config` directory.

The `./config` directory contains the following files:

- The OpenSSH server configuration file located in `ssh_host_keys/sshd_config` (auto-generated by the Docker container - do not edit it)
- The OpenSSH server's SSH keys located in `ssh_host_keys/`:
  Using the ECDSA algorithm:
  `ssh_host_ecdsa_key`

ssh_host_ecdsa_key.pub

Using the Ed25519 algorithm:

ssh_host_ed25519_key

ssh_host_ed25519_key.pub

Using the RSA algorithm:

ssh_host_rsa_key

ssh_host_rsa_key.pub

In a real-world scenario, you would have to generate your own keys using one algorithm and replace the ones provided by the Docker container. The keys provided by the Docker container are only for demonstration purposes.

- The authorized keys file located in .ssh/authorized_keys. This file contains the public keys that are allowed to connect to the SSH server. The file is empty by default. You will add your public key to this file later.

Connect to the SSH server with a password

Now that the SSH server is started, you can connect to it.

You can find the username and password in the openssh-server.env file.

In a terminal, run the following command, replacing <username> with the username found in the openssh-server.env file:

```
# Connect to the SSH server
ssh -p 2222 <username>@localhost
```

The -p option is used to specify the port to use. The default port is 22 but the SSH server is configured to use the port 2222.

The first time you connect to the SSH server, you will be asked to confirm the fingerprint of the SSH server. The fingerprint is displayed in the terminal.

The fingerprint is used to verify that the public key is the same as the one used to connect to the remote machine.

Accept the fingerprint by typing yes and pressing the Enter key.

The output should be similar to the following:

```
The authenticity of host '[localhost]:2222 ([::1]:2222)' can't be established.
ED25519 key fingerprint is SHA256:HiinVOAlUHtTTt6uoBcrSlBDbXiiEbvmY6N88Gb5atk.
```

```
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '[localhost]:2222' (ED25519) to the list of known hosts.
<username>@localhost's password:
```

Enter the password found in the `openssh-server.env` file and press the `Enter` key.

The output should be similar to the following:

```
Welcome to OpenSSH Server
87c8a04263d8:~$
```

Congratulations! You are now connected to the SSH server.

Create a file and list the files with the `ls` command:

```
# Create a file on the remote server
echo "This file is on the remote server" > remote.txt

# List the files
ls -la
```

You should see the `remote.txt` file in the `./config` directory of the Docker host as well.

You can exit the SSH server with the `exit` command.

Connect to the SSH server without a password

Using a password to connect to a remote machine is not recommended.

The reason is that a password can be guessed by brute force attacks, hard to remember and can be intercepted.

A key pair is more secure than a password.

In this section, you will generate a key pair and use it to connect to the SSH server.

Generate a key pair

In a terminal, run the following command:

```
# Generate a temporary key pair for demonstration purposes
ssh-keygen -t ed25519 -f demo_ed25519 -C "Demo"
```

The `-t` option is used to specify the algorithm to use. The `-f` option is used to specify the filename. The `-C` option is used to specify the comment.

The output should be similar to the following:

```
Generating public/private ed25519 key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in demo_ed25519
Your public key has been saved in demo_ed25519.pub
The key fingerprint is:
SHA256:vywAG98bh3ROT/jraMQGlZAeQYW0dSdHVTyEXAyhGnk
The key's randomart image is:
+--[ED25519 256]--+
|       o=*o.++@=+|
|        +o+..* o.|
|       ..= E    .|
|    o   + B .    |
|     = oSO +     |
|    . o +.* o    |
|       . *.  .   |
|        o..o.    |
|         o+..    |
+----[SHA256]-----+
```

You should now have two files in the current directory:

- `demo_ed25519`: the private key
- `demo_ed25519.pub`: the public key

Transfer the public key to the SSH server

In order to use the key pair to connect to the SSH server, you need to transfer the public key to the SSH server.

In a terminal, run the following command, replacing `<username>` with the username found in the `openssh-server.env` file:

```
# Transfer the public key to the SSH server
ssh-copy-id -i demo_ed25519 -p 2222 <username>@localhost
```

The -i option is used to specify the public key to use.

Enter the password found in the openssh-server.env file and press the Enter key.

The output should be similar to the following:

```
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "demo_ed25519.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
<username>@localhost's password:

Number of key(s) added:        1

Now try logging into the machine, with:   "ssh -p '2222' '<username>@localhost'"
and check to make sure that only the key(s) you wanted were added.
```

The public key should now be transferred to the SSH server.

You might have noticed that you used the private key in the command line but it was able to find and transfer the public key automatically.

You can check the ./config/.ssh/authorized_keys file to verify that the public key has been added to the file.

Connect to the SSH server without a password

You can now connect to the SSH server without a password.

In a terminal, run the following command, replacing <username> with the username found in the openssh-server.env file:

```
# Connect to the SSH server
ssh -i demo_ed25519 -p 2222 <username>@localhost
```

If you entered a passphrase when generating the key pair, you will be asked to enter it to decrypt the private key.

In order to avoid entering the passphrase every time you connect to the SSH server, you can use the ssh-agent command to store the passphrase. GitHub has a good documentation regarding this point: https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent#adding-your-ssh-key-to-the-ssh-agent.

SSH keys are greatly used with automation processes such as CI/CD pipelines, deployment scripts, etc. In this case, the passphrase is not required as the private key is not used by a human and should be protected by other means.

Congratulations! You are now connected to the SSH server without a password, relying on a key pair.

This is more secure but you can go even further by deactivating the password authentication on the SSH server. This is a good practice to avoid brute force attacks.

Deactivate the password authentication on the SSH server

In order to deactivate the password authentication on the SSH server, you need to stop the SSH server and edit the openssh-server.env file.

This will automatically edit the ./config/ssh_host_keys/sshd_config configuration file.

Stop the SSH server with Ctrl + C and edit the openssh-server.env file.

In the openssh-server.env file, change the PASSWORD_ACCESS variable value from true to false.

Start the SSH server again with the docker compose up command.

Try to access the SSH server with a password. You should get an error message saying that the password authentication is disabled.

You can check the ./config/ssh_host_keys/sshd_config configuration file to verify that the PasswordAuthentication option is set to no.

Add a second public key to the SSH server

For demonstration purposes, you will add a second public key to the SSH server. You will then revoke it in the next section to remove the access to the SSH server to this key.

Generate a second key pair with the ssh-keygen command called revoke_ed25519.

Transfer the public key to the SSH server with the ssh-copy-id command.

Tip

The ssh-copy-id must use the previous private key to connect to the SSH server in order to add the new public key to the authorized_keys file.

You can specify the private key to use with the -o option:

```
# Transfer the public key to the SSH server
ssh-copy-id -i revoke_ed25519  -o 'IdentityFile demo_ed25519' -f -p 2222
             <username>@localhost
```

Try to access the SSH server with the new public key. You should be able to connect to the SSH server.

The authorized_keys file should now contain two public keys.

You can log out of the SSH server with the exit command.

Revoke the second public key from the SSH server

Connect to the SSH server with the first public key.

You now must open an editor to remove the second public key from the authorized_keys file.

You can use the Vi editor or the one you are the most comfortable with.

```
# Open the authorized_keys file with the vi editor
vi ~/.ssh/authorized_keys
```

Use the arrow keys to move the cursor up and down.

Remove the second public key from the authorized_keys file by moving the cursor to the public key and pressing the d key twice.

You can check the value of the comment field to identify the public key to remove or the hash of the public key.

Save the file by typing :wq (write and quit).

Congrats! You were able to exit the Vi editor! You will not be trapped forever!

If you are interested in learning more about the Vi/Vim editor, you can check the following resource: https://github.com/iggredible/Learn-Vim/tree/master. Learning to use a terminal-based editor such as Vim or Nano is a good skill to have to be able to edit files on a remote machine without a graphical interface.

You can check the authorized_keys file to verify that the second public key has been removed:

```
# List the authorized_keys file
cat ~/.ssh/authorized_keys
```

You should now only see the first public key.

Now try to access the SSH server with the second public key. You should get an error message saying that the public key is not allowed to connect to the SSH server.

Congratulations! You have successfully revoked the second public key from the SSH server.

Managing access to a remote machine with public keys is easy and more secure.

Revoking a public key is as easy as removing it from the authorized_keys file.

Copy files with SCP

In this section, you will learn how to copy files from/to a remote machine with SCP.

Copying files with SCP is similar to copying files with the cp command.

Create a file on the local machine

On the local machine, create a new file called local.txt:

```
# Create a file on the local machine
echo "This file is on the local machine" > local.txt
```

You should now have a local.txt file in the current directory and a remote.txt file on the remote server created in the previous section.

Copy the file from the local machine to the remote machine

In a terminal, run the following command, replacing <username> with the username found in the openssh-server.env file:

```
# Copy the file from the local machine to the remote machine
scp -i demo_ed25519 -P 2222 local.txt <username>@localhost:local.txt
```

The -P option is used to specify the port to use with SCP.

The syntax is similar to the cp command:

```
[username@source-ip:]source [username@destination-ip:]destination
```

The first local.txt is the file to copy from the local machine. The second local.txt is the file to copy to the remote machine, starting from the home directory of the user.

The output should be similar to the following:

```
local.txt          100%   34    30.2KB/s   00:00
```

Congratulations! The transfer was successful.

If you log on the remote machine, you should see the local.txt file in the home directory of the user with its content:

```
# Display the content of the local.txt file
cat local.txt
```

Copy a file from the remote machine to the local machine

To copy a file from the remote machine to the local machine, you can use the same command as before but with the source and destination inverted:

```
# Copy the file from the remote machine to the local machine
scp -i demo_ed25519 -P 2222 <username>@localhost:remote.txt remote.txt
```

The output should be similar to the following:

```
remote.txt        100%   34    25.8KB/s   00:00
```

Congratulations! The transfer was successful.

You should see the remote.txt file in the current directory with its content:

```
# Display the content of the remote.txt file
cat remote.txt
```

Go further

This is an optional section. Feel free to skip it if you do not have time.

- Are you able to deactivate the password authentication on the SSH server? The key authentication should still work. This is a good practice to avoid brute force attacks.

Conclusion

What did you do and learn?

In this chapter, you have had a refresh about security and learned how to use SSH to connect to a remote machine. You have also learned how to copy files from/to a remote machine with SCP.

SSH and SCP are very useful tools to connect to a remote machine and copy files from/to a remote machine. They are widely used in the industry.

With the help of OpenSSH Server and Docker, you have been able to experiment with SSH and SCP in a safe environment.

Test your knowledge

At this point, you should be able to answer the following questions:

- What is SSH ?
- How does SSH work?
- How to copy files from/to a remote machine with SCP?
- Why is it not recommended to use a password to connect to a remote machine?
- What is the difference between the private key and the public key?
- What is the difference between the private key and the passphrase?

Finished? Was it easy? Was it hard?

Can you let us know what was easy and what was difficult for you during this chapter?

This will help us to improve the course and adapt the content to your needs. If we notice some difficulties, we will come back to you to help you.

➡ GitHub Discussions

You can use reactions to express your opinion on a comment!

What will you do next?

In the next chapter, you will learn the following topics:

- Java TCP programming
    - How to send an email with Java
    - How to create a TCP server
    - How to create a TCP client
    - How to handle multiple clients with concurrency

Additional resources

Resources are here to help you. They are not mandatory to read.

- *None yet*

Missing item in the list? Feel free to open a pull request to add it!

Sources

- Main illustration by Mathew Schwartz on Unsplash