SSH: key of success

Protocol & Applications

Philippe.Dos-Santos@ipsl.fr

Sebastien.Gardoll@ipsl.fr

Philippe.Weill@latmos.ipsl.fr

Table of content

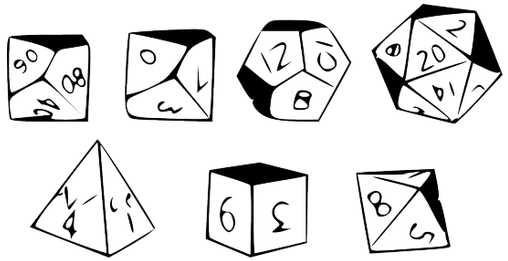
INTRODUCTION

Secure Shell

- Protocol: secure connection specification (encryption)
- Applications
  - Connection to a remote host shell: `ssh` (replace telnet, FTP, R-command, etc.)
  - File management: transfer, delete, move and synchronize files & directories (`scp`, `sftp`, `rsync`).
  - Tunneling: make services accessible and secured.

BASIC CRYPTOGRAPHY KNOWLEDGE

Fundamental functions



- Random number generator

Generate number from random sources : mouse gesture, keyboard entry, /dev/[u]random, etc.

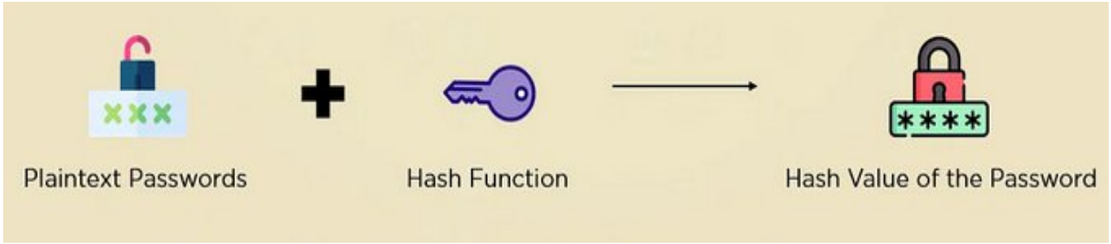
- Hash/Digest

Algorithm based on mathematical functions that map arbitrary size data to fixed size set of values.

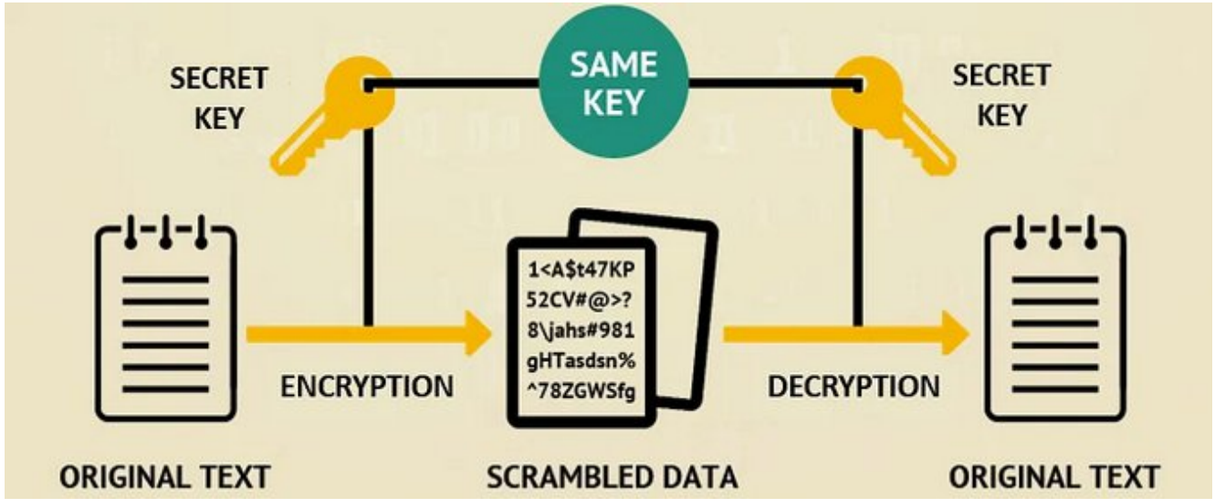
- Functions are irreversible: it is not possible to compute the original data from the hash values.
- Use cases: password hashing (with salt), file integrity (transfer), electronic signature, etc.
- Algorithms: md5, sha1, sha256, sha512, sha3, etc.
- Command lines: md5sum sha1sum sha256sum sha512sum, etc.

```
user@host:~$ sha1sum .bashrc
8976565afec640267e044381ff87163e79bcb895 .bashrc
```

- Symmetric, asymmetric, hybrid key cryptography.

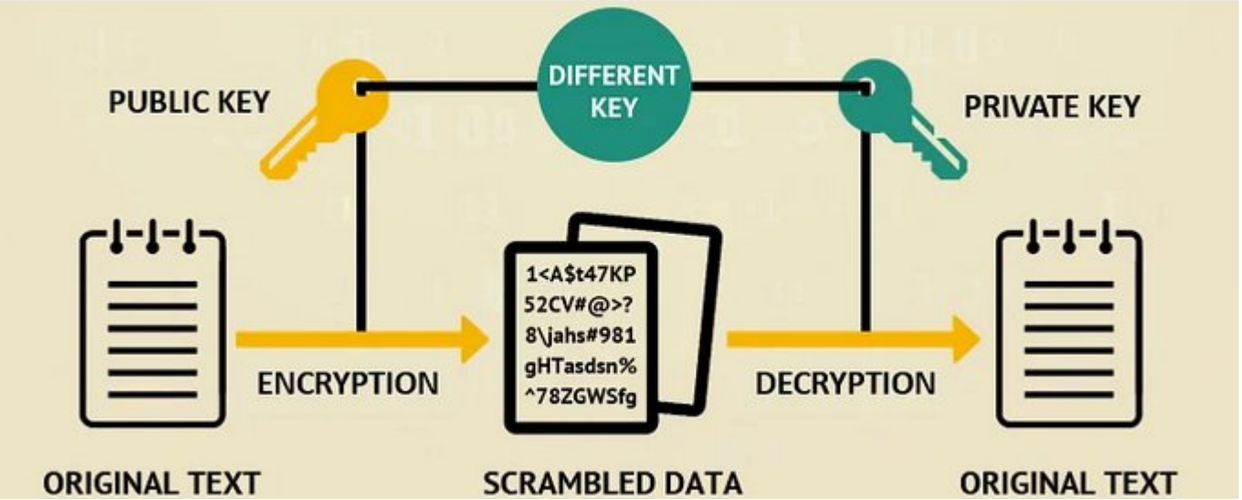


Symmetric key cryptography



- Principle: encrypt/decrypt data with the same key.
- Synonyms: shared key, secret key, session key, etc.
- Ciphers: DES, 3DES, AES, BLOWFISH, IDEA, etc.
- Advantages: very fast, high entropy, etc.
- Problem: how to securely share the key?

Asymmetric key cryptography



- Principle: with a couple of key, when encryption is made by one key, decryption could be only made by the other key and vice versa.
- Ciphers: RSA (keys <= 2048 bits deprecated), DSA (deprecated), ECDSA, ED25519 (recommended), etc.
- Notion: private and public keys, just a matter of visibility, they are not different! You can encrypt with a private key and decrypt with a public key (it's the principle of digital signatures using hash functions)
- Advantage: sharing the encryption key is no longer a problem.
- Problem: It's slow, 1000 times slower than symmetric.

Hybrid cryptography

- Principle: a symmetric key, called session key, is exchanged using an asymmetric cryptography.
- Data encryption during the transmission is made with the session key.
- Well known applications use hybrid cryptography: SSH, VPN, SSL/TLS (HTTPS, IMAPS, POPS, SMTPS, etc.)

CONNECTION & AUTHENTICATION

Establishing a secure connection

SSH Client

SSH Server

1. Client ask for server's public key and supported SSH protocol version & symmetric ciphers.

2. Client chooses a symmetric cipher and generate a key for it: the session key.

3. Client sends the session key encrypted with the server's public key.

4. Client informs the server about the chosen symmetric cipher.

5. Server decrypt the session key with its private key.

6. Client and Server encrypt/decrypt information using the session key.

Host authenticity checking 1/2

Remote host checking depends on the value of option `StrictHostKeyChecking` in `~/.ssh/config`: `no`, `yes` or `ask`.

If `StrictHostKeyChecking` is set to `no`:

```
~$ ssh user@machine
Warning: Permanently added 'machine,10.0.0.1' (RSA) to the list of known hosts.
Password:
```

If `StrictHostKeyChecking` is set to `yes`:

```
~$ ssh user@machine
No RSA host key is known for machine and you have requested strict checking.
Host key verification failed.
```

If `StrictHostKeyChecking` is set to `ask`:

```
~$ ssh user@machine
The authenticity of host 'machine (10.0.0.1)' can't be established.
RSA key fingerprint is 8a:1f:39:4b:a8:d0:13:9a:cf:c3:c2:13:2d:42:9f:b0.
Are you sure you want to continue connecting (yes/no)?
```

Host authenticity checking 2/2

When the server's public key has changed, `ssh` warns you about a possible man in the middle attack:

```
~$ ssh user@machine
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that the RSA host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
55:3b:ab:11:8f:96:80:a6:9c:c5:63:5f:ed:87:bc:94.
Please contact your system administrator.
Add correct host key in /home/user/.ssh/known_hosts to get rid of this message.
Offending key in /home/user/.ssh/known_hosts:1
Password authentication is disabled to avoid man-in-the-middle attacks.
Keyboard-interactive authentication is disabled to avoid man-in-the-middle attacks.
Agent forwarding is disabled to avoid man-in-the-middle attacks.
X11 forwarding is disabled to avoid man-in-the-middle attacks.
Password:
```

User authentication methods

As a secure connection between client and server is established (first step), access is granted by:

- Password: the client sent his password via secure connection and the server check if password is valid.
- Asymmetric keys, based on challenge-response protocol: If authentication by key is enable, the server create a challenge encrypted by the registered client public key (like in first step). If client passes the challenge (decrypt with its private key), the server grant access to the client.
- Host: system using hosts.equiv and known_host defined by admin. As you're already authenticated on one host, you can access to the others (used sometime inside HPC clusters).
- Kerberos, smartcard (YubiKey), PAM, certificate, 2 factors (YubiKey), etc.

SSH client: ssh 1/2

- Remote host connection

Syntax: `ssh [username@]<hostname FQDN>`

Example for user user on spirit1:

```
user@port-500:~$ ssh user@ssh-gw.ipsl.fr # or ssh ssh-gw.ipsl.fr
Password:
user@ssh-gw:~$
```

- Command line or remote script executed on remote host

Syntax: `ssh [username@]<hostname FQDN> "<command line or script absolute path>"`

Example of files and directories listing the home directory of user user on spirit1:

```
user@port-500:~$ ssh user@ssh-gw.ipsl.fr "ls -l"
Password:
total 102276
drwx------ 2 user ipsl        44 juin  28  2022 bin
drwxr-xr-x 3 user ipsl        27 mars   2  2023 data
drwxr-xr-x 4 user ipsl       243 avril 21 09:45 tmp
```

SSH client: ssh 2/2

- Local script executed on remote host

Syntax: `ssh [username@]<hostname FQDN> "bash -s" < /path/to/my/script`

Example of files and directories listing the home directory of user user on spirit1:

```
user@port-500:~$ echo "ls" > tmp.sh # tmp.sh only exists on localhost (port-500)!
user@port-500:~$ ssh user@ssh-gw.ipsl.fr "bash -s" < tmp.sh
Password:
bin
data
tmp
```

Useful for scripts factorization: one script for multiple hosts!

IPSL SSH jump hosts

- IPSL Fédération: ssh-gw.ipsl.fr
- LATMOS:
  - Guyancourt: cartman.latmos.ipsl.fr
  - Jussieu: sirocco.latmos.ipsl.fr
- LISA
- LMD: ssh-in.lmd.jussieu.fr
- LOCEAN: cerbere.locean-ipsl.upmc.fr
- LSCE
- IPSL cluster head nodes:
  - Spirit: spirit1.ipsl.fr & spirit2.ipsl.fr
  - SpiritX: spiritx1.ipsl.fr & spiritx2.ipsl.fr
  - HAL: hal.ipsl.fr

CLIENT CONFIGURATION

Configuration files

User configuration files are located in the `~/.ssh` directory (whereas admin files are located in `/etc/sshd` and are not intended for users):

- Client configuration files:
  - config: a set of instructions that configure the SSH commands (`ssh`, `scp`, `sftp`, etc.)
  - known_hosts: a set of server's public keys already checked/approved (seen in host authenticity).
  - public and private key files like: id_ed25519 & id_ed25519.pub, id_rsa & id_rsa.pub, etc. (will see their creation).

- Server configuration file:
  - authorized_keys: a set of client's public keys that are authorized for the user authentication (will see in authentication by key pair).

- config, know_hosts and authorized_keys files are not created by default. You may have to create them.

config 1/2

It contains the list of SSH client instructions.

Typical instructions applied for all remote hosts (*):

```
Host *
    StrictHostKeyChecking ask # Specifies the host authenticity checking behavior.
    ServerAliveInterval 90s   # Prevents closing an idle connection during 90 seconds.
    ForwardX11 yes            # Enable X-Forwarding, the window system (-X option).
    ForwardAgent yes          # Grant remote hosts to use local keys without deploying them.
```

config 2/2

doc

- Connection configuration

Defining a connection to spirit1 for user user:

```
Host spirit1
    HostName spirit1.ipsl.fr
    User user
    IdentityFile ~/.ssh/ipsl_id_ed25519
```

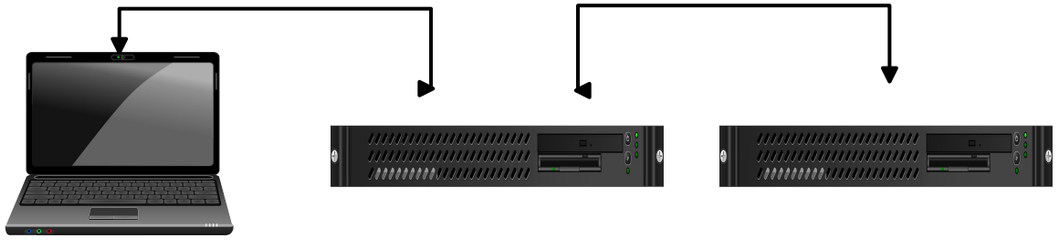
```
user@port-500:~$ ssh spirit1
user@spirit1:~$
```

- Jumping connection configuration

Defining a connection to jean-zay through spirit1 for user user:

```
Host jean-zay
    HostName jean-zay.idris.fr
    User uuf18sd
    IdentityFile ~/.ssh/jz_id_ed25519
    ProxyCommand ssh spirit1 -W %h:%p
```

```
user@port-500:~$ ssh jean-zay
uuf18sd@jean-zay1:~$
```



authorized_keys & known_hosts

- Example of authorized_keys content

client's public key cipher

client's public key

```
ssh-ed25519 AAAAC3NzaD1lZDI1NTE6BAAAIBtuYLxUq8waHr0oxM+1/PNu0haBzesl+IdPtrm8hfT
user@organization.com
```

- Example of known_hosts content

server FQDN

server's public key cipher

```
spirit1.ipsl.fr ssh-ed25519
AAAAC3NzaC1lZDI1NTE5AAAAIN0EP/A5FTncSzXPy9PyDblFy5KnKjMOyaKMeuLS6hhP
```

server's public key

SSH KEY MANAGEMENT

Key based authentication

- Strong authentication based on:
  - Challenge-response protocol.
  - Strong asymmetric ciphers (ECDSA, Ed25519, etc.)
- The private key remains on client-side and never leave it, except for backup (example: on SSH jump hosts).
- The private key is encrypted to prevent any unwanted used. Decrypted private keys can be hang on key rings to save further asking password.
- The public key is deployed on the server-sides, manually (editing authorized_keys) or automatically (`ssh-copy-id`).
- Keys can used from and to any machines: SSH is widely supported in Unix world.
- After a bit of work (deploying the public key and hunging decrypted private key on a key ring), you will be able to connect to a server without asking password, as long as the key ring is running (login session).

Key pair generation

doc

- Ed25519 keys generation example

```
user@port-500:~$ ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/user/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/user/.ssh/id_ed25519
Your public key has been saved in /home/user/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:jbk2NXPeLXCjwijozGtuWVvxAxyymDAR5ik35MAnV1k user@port-500
The key's randomart image is:
+--[ED25519 256]--+
|o =o..oE         |
| Oo+ .. .        |
|. Xo o + .       |
| o .o . ++       |
|        S+= o o  |
|     .. .=o= = o |
|    .o.o= o.o o .|
|   ++ .o . .   . |
|   +*.           |
+----[SHA256]-----+
```

- Key file permissions

```
user@port-500:~$ $ ls -l ~/.ssh
total 16
-rw-------@ 1 user  root  411 11 sep 14:43 id_ed25519
-rw-r--r--@ 1 user  root  102 11 sep 14:43 id_ed25519.pub
```

- .ssh and home permissions: never `chmod 777`!

Private key password

- The private key is encrypted (symmetric cipher), therefore the associated password:
  - Must be at least 14 characters long. Keepass and passphrase should be considered.
  - Must never be trivial (single word in any existing language).
  - Must never be your email, user account or your login session password.
  - Should include special characters (#, }, $, *, §, etc.) But mind your keyboard layout!

- Keepass password manager:
  - Is an encrypted data bases of logins, passwords, URL and notes. Easily copy & past your login names and passwords.
  - Is granted by ANSSI.
  - Can generate:
    - Passwords with as many characters as you like.
    - Passphrases with as many words as you like.
  - Display password and passphrases entropy.

Public key deployment: manual

Copy the public key and paste it into the remote authorized_keys file.

- On client-side 

```
user@port-500:~$ scp ~/.ssh/id_ed25519.pub user@ssh-gw.ipsl.fr:./.ssh
```

Or read the content of the public key file and copy its contents (`vi`, `less`, `more`, `cat`, etc.)

- On server-side 

```
user@ssh-gw:~$ cat ~/.ssh/id_ed25519.pub >> ~/.ssh/authorized_keys
user@ssh-gw:~$ chmod 600 ~/.ssh/authorized_keys
```

Or edit the authorized_keys file and paste the public key (`vi`, `nano`, etc.)

The data center of the IPSL doesn't authorize the modification of the authorized_keys file content on its clusters (Spirit, SpiritX and HAL). You may contact meso-support@ipsl.fr for any key access modifications.

Public key deployment: automatic (recommended)

- On client-side

Syntax: `ssh-copy-id -i <path to the public key> [username@]<hostname FQDN>`

Example:

```
user@port-500:~$ ssh-copy-id -i ~/.ssh/id_ed25519.pub user@ssh-gw.ipsl.fr
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: ".ssh/id_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
(user@ssh-gw.ipsl.fr) Password:

Number of key(s) added:        1

Now try logging into the machine, with:   "ssh 'user@ssh-gw.ipsl.fr'"
and check to make sure that only the key(s) you wanted were added.
```

Key backup

doc

- Backup your SSH keys:
  - Prevent loosing your access from hardware failures or human errors.
  - Carefully copy your SSH keys, especially the private key: never decrypted!
  - We recommend to copy your SSH keys on the SSH jump host of your lab. Jump hosts are still accessible with password (not the clusters of IPSL).
  - Do not change the permissions of the private key, keep it not readable for group and other (`chmod 600`). Whereas the public key has to be readable for all users (`chmod 644`).

Key rings

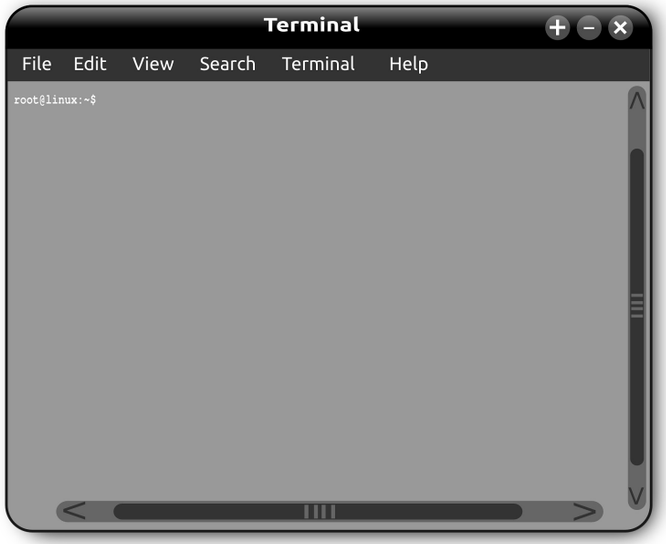
- Principle
  - You decrypt the private keys once per login session, then use them as many times as you like without having to decrypt them each time they are used.
  - Decrypted private keys are flushed from key rings when user logs off. Private keys stay decrypted as long as login session lives.
- Shell key ring: ssh-agent (available on all OS).
- Graphical user interface key rings:
  - MacOSX: Keychain.
  - Linux & Windows Subsystem for Linux: Gnome keyring, KDE wallet, etc.
  - Windows: Mobaxterm (much more than a key ring!), etc.

ssh-agent (recommended on Linux)

`ssh-agent` is a shell key ring. First you have to activate it (if not), only once per login session, then you can add decrypted private keys as many as you want with `ssh-add`.

- Activate ssh-agent only once per login session (should be added in ~/.bashrc)

```
user@port-500:~$ eval $(ssh-agent)
Agent pid 2317917
```



- Decrypt a private key and hang it up on `ssh-agent`

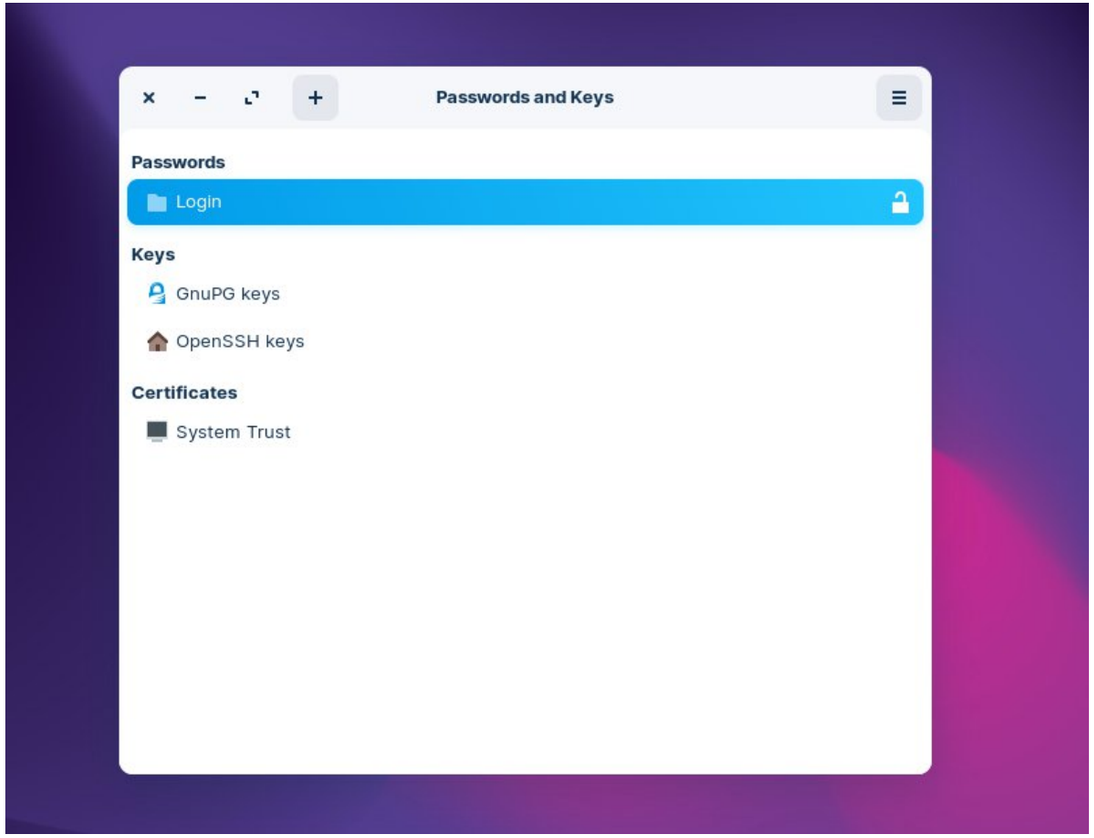
Syntax: `ssh-add <path to the private key>`

Example:

```
user@port-500:~$ ssh-add ~/.ssh/id_25519
Enter passphrase for /home/user/.ssh/id_ed25519:
Identity added: /home/user/.ssh/id_ed25519 (user@ipsl.fr)
```

Gnome keyring/Seahorse

Gnome keyring is the default key ring system on Gnome desktop. Seahorse is one of its graphical user interface and makes it easier to use. As for ssh-agent, the decrypted keys can last as long as the user session is running.



MacOSX Keychain

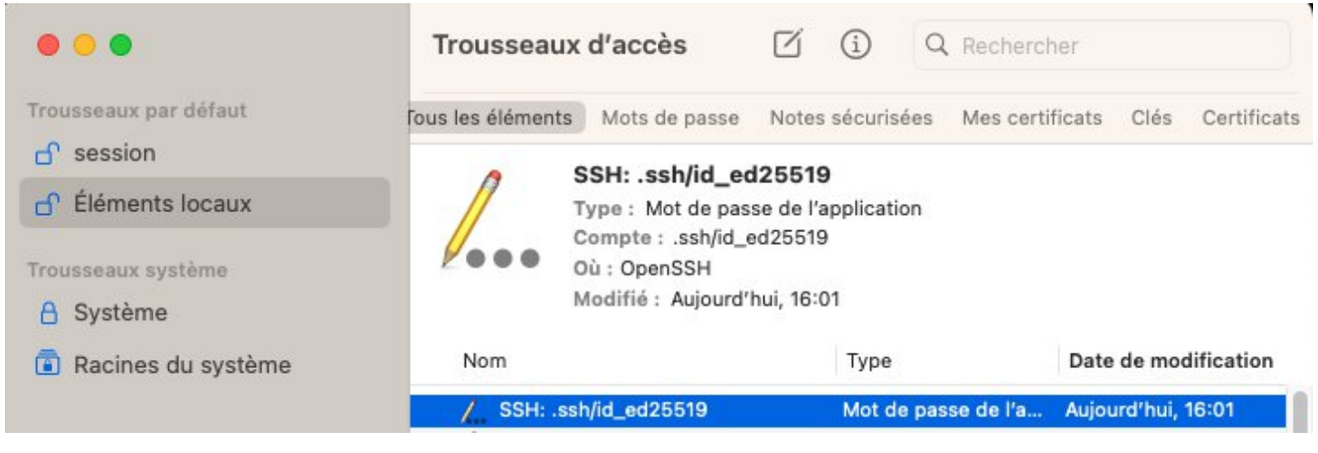
Keychain is the native key ring on MacOSX. It doesn't accept decrypted private keys, but it can store their password.

- Decrypt a private key, hang it up on `ssh-agent` and store the password of the key on Keychain

Syntax: `ssh-add --apple-use-keychain <path to the private key>`

Example:

```
user@port-500:~$ ssh-add --apple-use-keychain ~/.ssh/id_ed25519
Enter passphrase for .ssh/id_ed25519:
Identity added: .ssh/id_ed25519 (user@ipsl.fr)
```



FILE APPLICATIONS

Secure Copy Protocol

- Scheme

```
scp [-r] sources target
```

`-r` stands for recursive: it enables copy of directories and its contents (files and subdirectories).

- Send local files to remote host

Syntax:

```
scp <path to the file> [path to the file] [username@]<hostname FQDN>:<destination path>
```

Example of copying the local command `more` into the home directory on spirit1:

```
user@port-500:~$ scp /usr/bin/more user@spirit1.ipsl.fr:/home/user # Absolute path.
user@port-500:~$ scp /usr/bin/more user@spirit1.ipsl.fr:. # Relative path.
```

- Send remote files to local host

Syntax:

```
scp [username@]<hostname FQDN>:<path to the file> <local parent directory path>
```

Example of copying the command `less` on spirit1 into the local home directory:

```
user@port-500:~$ scp user@spirit1.ipsl.fr:/usr/bin/less ~/
```

- Working with configured connection (~/.ssh/config)

```
user@port-500:~$ scp -c aes128-ctr -r /some/data jean-zay:/gpfsscratch/rech/epp/xxx31d
```

SSH File Transfer Program 1/4

- Connection to the SFTP commands interpreter

Syntax: `sftp [-r] [username@]<hostname FQDN>[:remote working directory path]`

`-r` stands for recursive: it enables copy entire directories when uploading and downloading.

- Download commands: download remote files into the local working directory

```
get [-R] <path on remote host to the file> # to start downloading a file or directory (-R)
reget [-R] <path on remote host to the same file> # to resume an interrupted download.
```

- Upload commands: upload local files into the remote working directory

```
put [-R] <path on local host to the file> # to start uploading a file or directory (-R)
reput [-R] <path on local host to the same file> # to resume an interrupted upload.
```

`put` et `reput` don't create remote directory: you must create it before uploading data (`mkdir`)!

- Quit SFTP interpreter: `quit` or `exit` or keyboard shortcut: CTRL + d (like any shell)

- Other commands: `rm`, `copy`, etc. (`ssh` is recommended when working with directories.)

- Navigating in local and remote file system:
  - Remote: `cd`, `ls`, `pwd`, `mkdir`, etc.
  - Local: `lcd`, `lls`, `lpwd`, `lmkdir`, etc.

SSH File Transfer Program 2/4

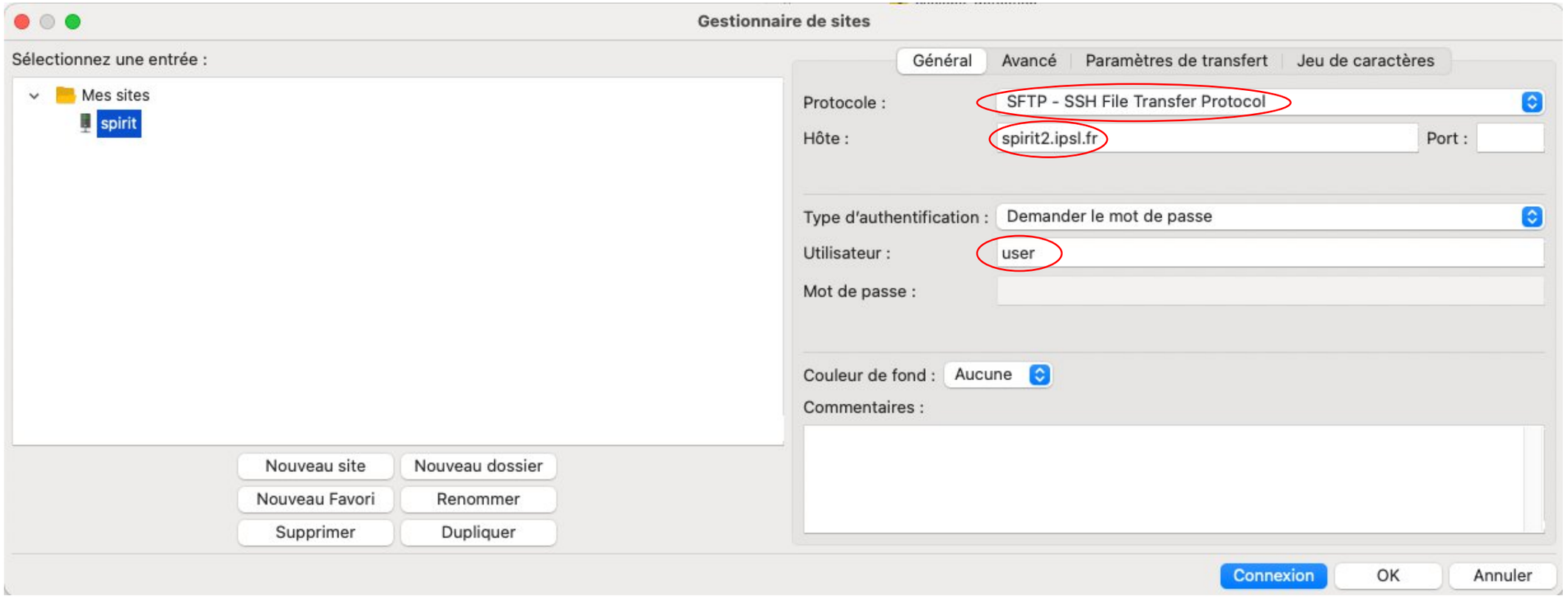
- Example: egress data on Jean-Zay with configured connection (~/.ssh/config). spirit1 is the registered machine to access Jean Zay.

```
user@port-500:~$ ssh spirit1
user@spirit1:~$ sftp -r jean-zay:/gpfsscratch/rech/epp/xxx31d
Connected to jean-zay.
Changing to: /gpfsscratch/rech/epp/xxx31d
sftp> mkdir some_data # Fix error: "Couldn't canonicalize: No such file or directory"
sftp> put -R /scratchu/user/some_data # -R is overkill with -r
Uploading /scratchu/user/some_data/ to /gpfsssd/scratch/rech/epp/xxx31d/some_data
Entering /scratchu/user/some_data/
Entering /scratchu/user/some_data/subdir
/scratchu/user/some_data/subdir/file1 100%  708KB   5.4MB/s   00:00
sftp> quit
user@spirit1:~$
```

SSH File Transfer Program 3/4

- Filezilla: free SFTP GUI

1. Set up a connection with site manager (file menu)



SSH File Transfer Program 4/4

2. Then drag and drop files and directories from a view to another.

Local file system view

Remote file system view

Rsync over SSH

- Rsync over ssh scheme

```
rsync <rsync options> <source_expression> <destination expression>
```

- Synchronizing a local directory on remote host

```
rsync --archive --hard-links --delete --human-readable --verbose --progress
/some/where/on/local_host/some_data/
[username@]<hostname FQDN>:/some/where/on/remote_host/some_data
```

Is equivalent to:

```
rsync --archive --hard-links --delete --human-readable --verbose --progress
/some/where/on/local_host/some_data [username@]<hostname FQDN>:/some/where/on/remote_host
```

- Example with configured connection (~/.ssh/config):

```
user@port-500:~$ rsync -a -H --delete -h -v --progress ~/some_data/ spirit1:/scratchu/user/some_data
```

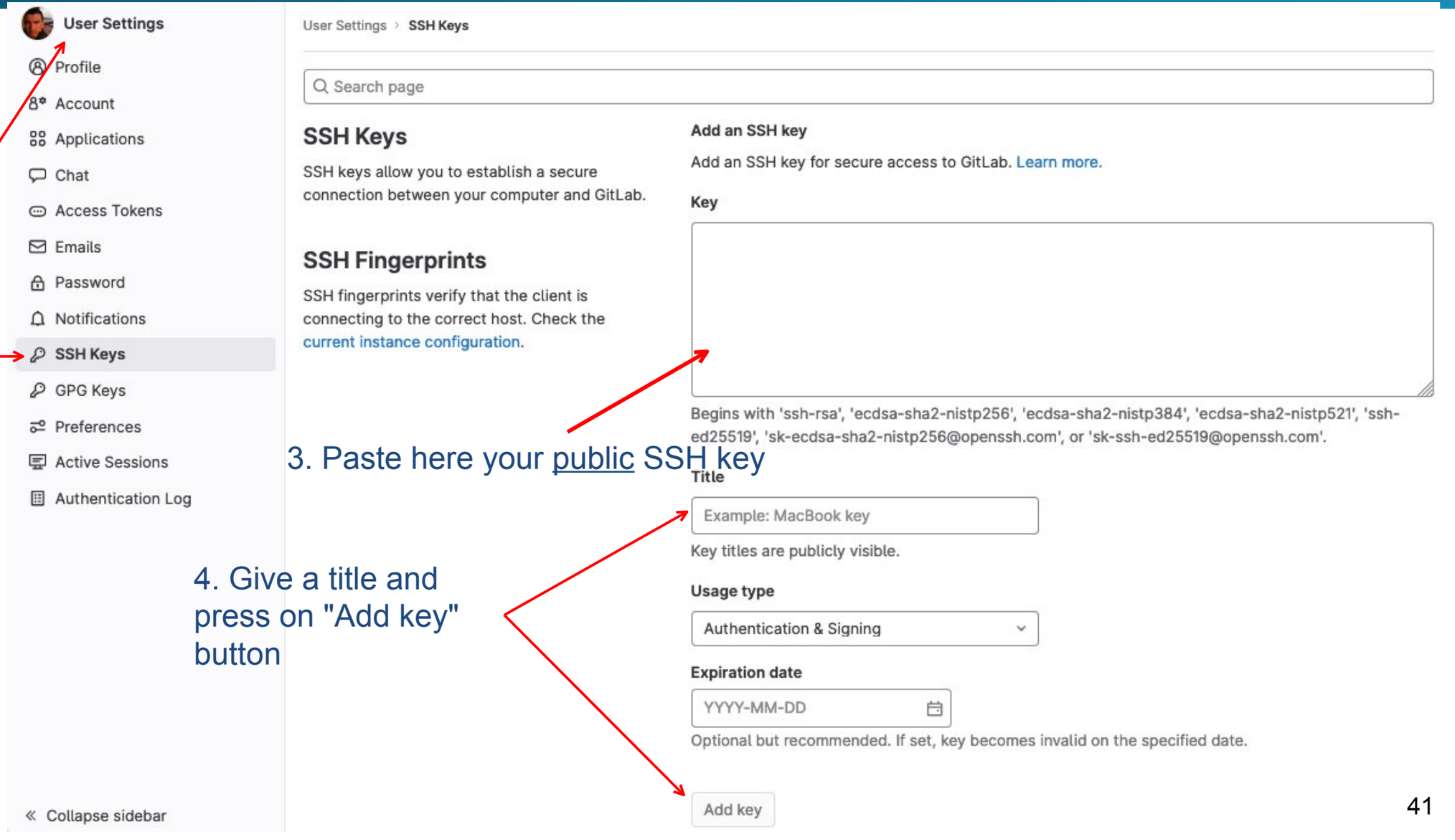
- Synchronizing a remote directory on local host: just invert source and destination!

Gitlab over SSH 1/2
Public key registration

1. Open your user settings menu

2. Select "SSH Keys" item

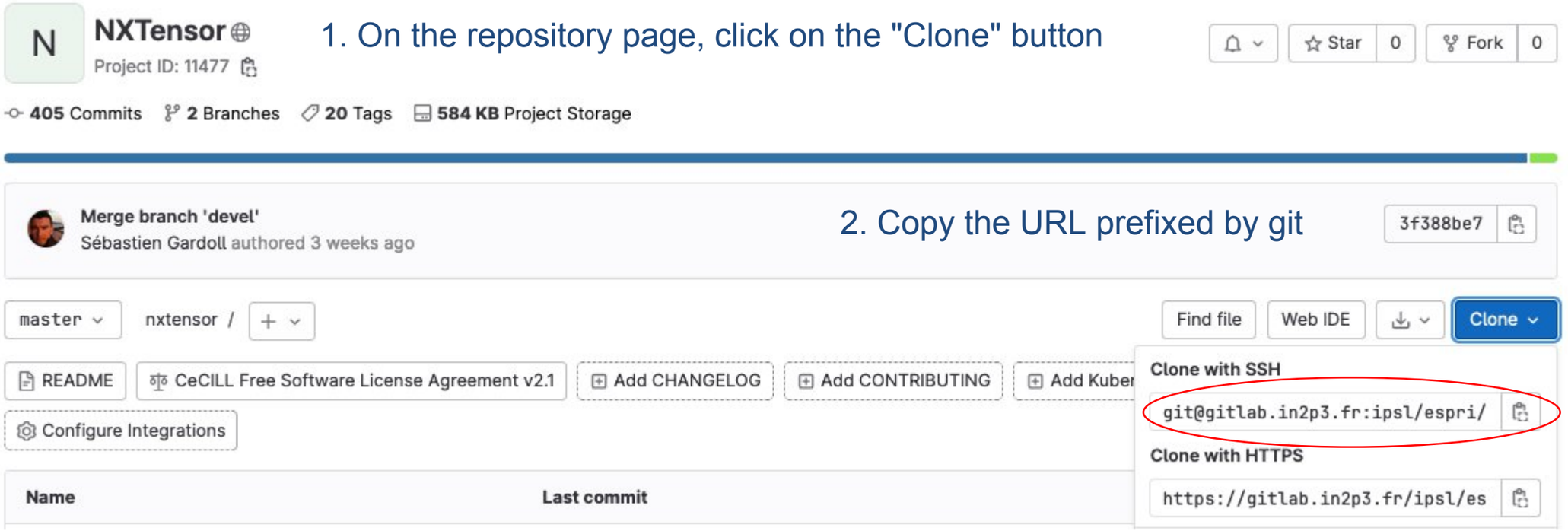
The public key
is added once and for all



3. Paste here your public SSH key

4. Give a title and press on "Add key" button

Gitlab over SSH 2/2
Cloning example



3. Finally, open a shell and clone the repository

```
user@port-500:~$ git clone git@gitlab.in2p3.fr:ipsl/espri/espri-ia/projects/nxtensor.git
```

`git pull` and `git push` will use the registered public key for authentication challenge: no more password asked during the login session if your private key is hung decrypted on a key ring. Github also supports the same feature.

SSH TUNNELING

Use cases

- Principle: encapsulate network communication in an SSH connection.
- Use cases:
  - Securing a connection (transport layer) but not the authentication. Example: VNC (remote desktop).
  - Access to services in restricted networks. Example: Web interfaces like JupyterLab, Tensorboard, Dask dashboard, etc. when running on IPSL clusters.

TCP & UDP ports

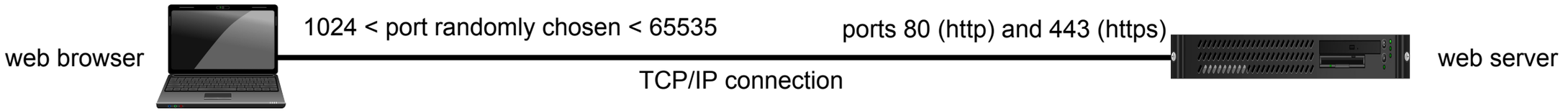
- Principle

Each network connection have two end points. A port refers to a TCP or UDP connection end point and is identified by a unique number (unsigned 16 bits integer). When two processes establish a connection, each process open its own port.

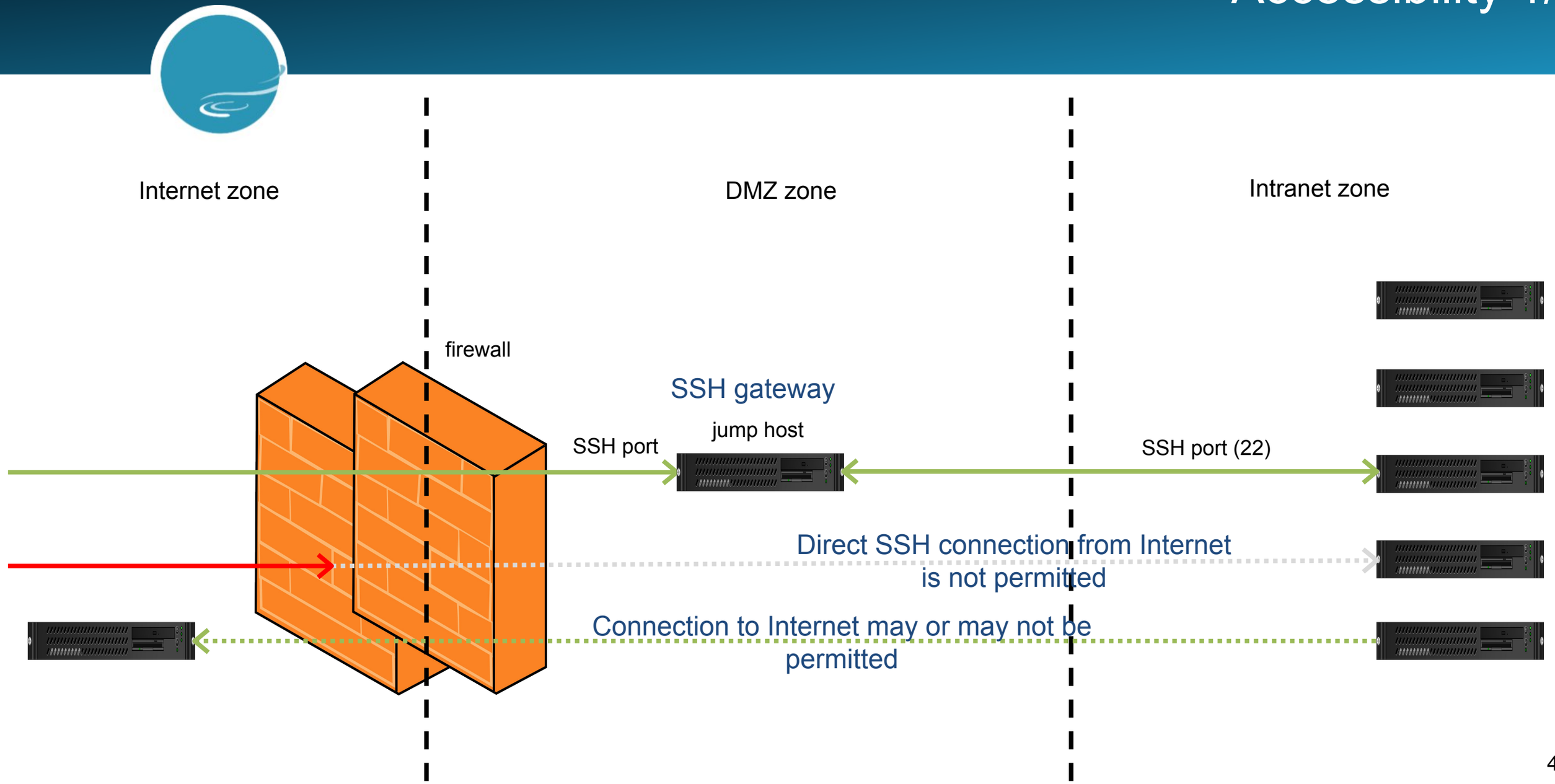
- Reserved port numbers:
  - Only root can open port numbers <= 1024.
  - Some port numbers are dedicated to specific protocols (80 HTTP, 443 HTTPS, 21 FTP, 22 SSH, etc.).
  - Services that implement these protocols, are expected to open the associated port numbers (on their side).

- Router and firewall can filter connections based on the port number, mac address (NIC id), etc. <=> Access to services is regulated according to a security policy. Usually, by default, nothing is permitted: intranet machines are not accessible from Internet, whatever the port.

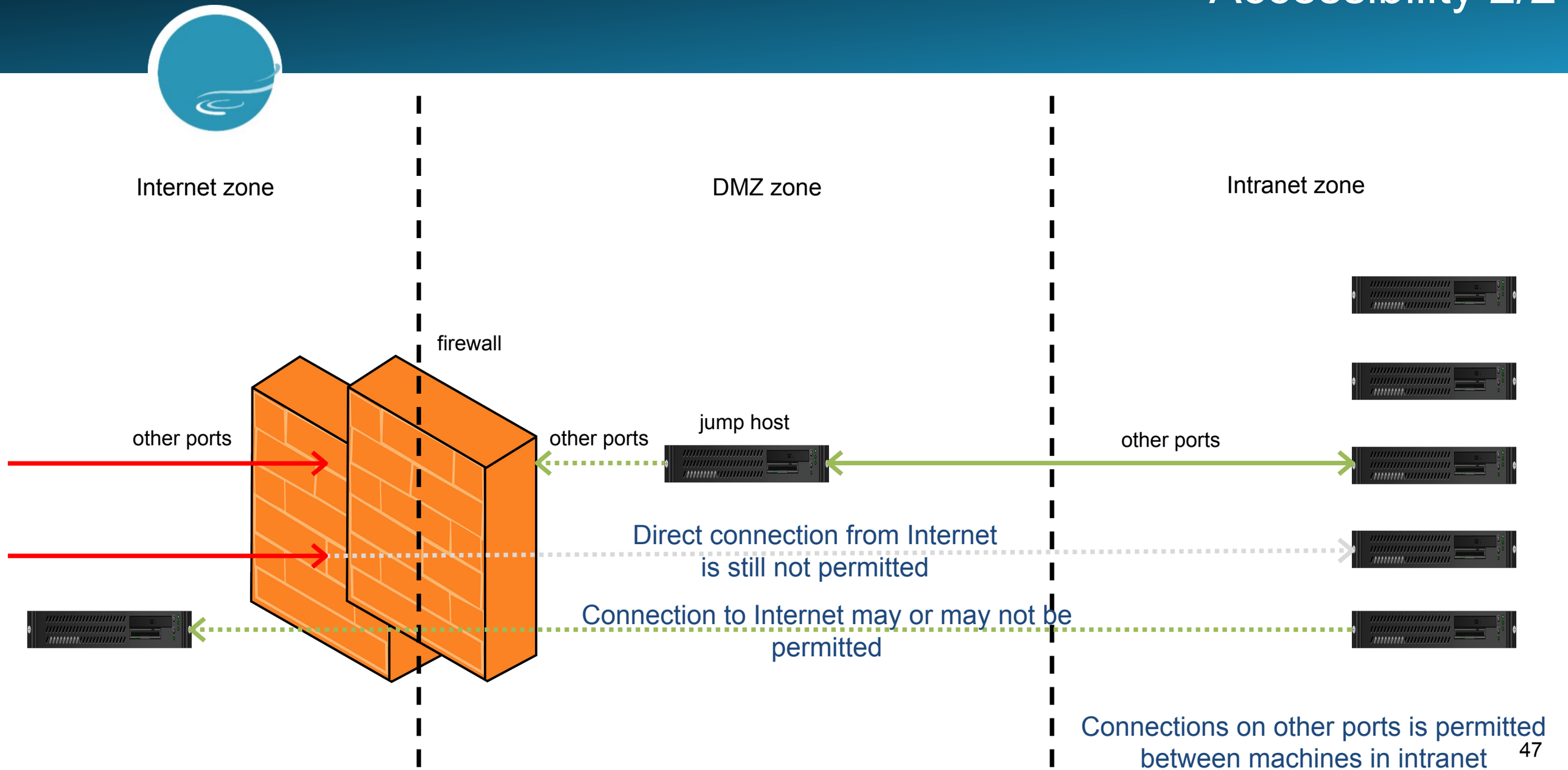
- Example of a web browser opening a page:



Accessibility 1/2



Accessibility 2/2

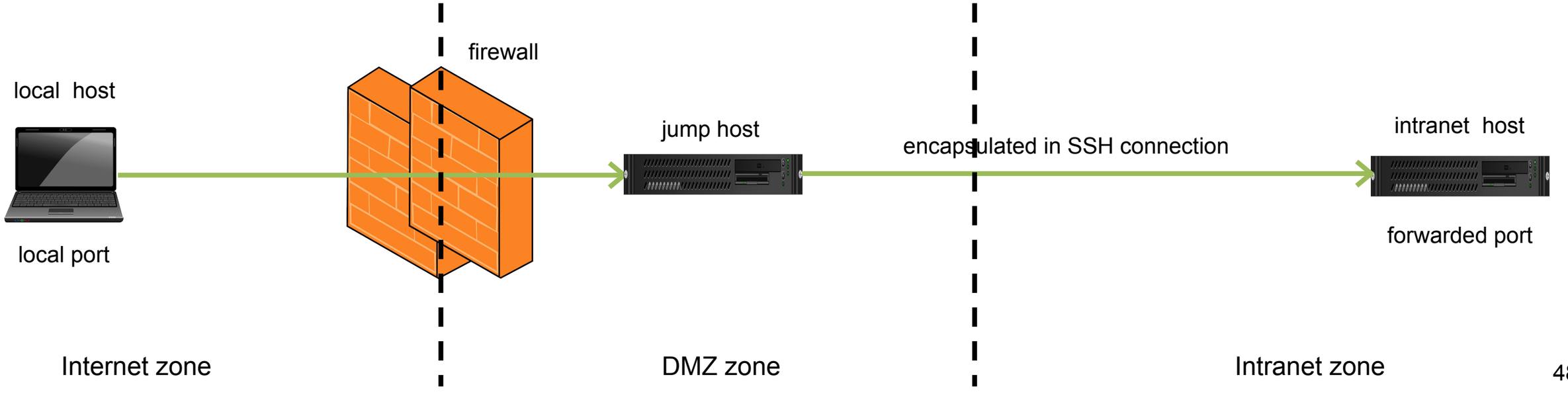


Local port forwarding

- Principle
  - Relay a port on an Internet host to an intranet host, through a jump host.
  - Any service that opens the forwarded local port on intranet host, is accessible on the Internet host. In other words, Internet host is able to access to services that run on intranet host.

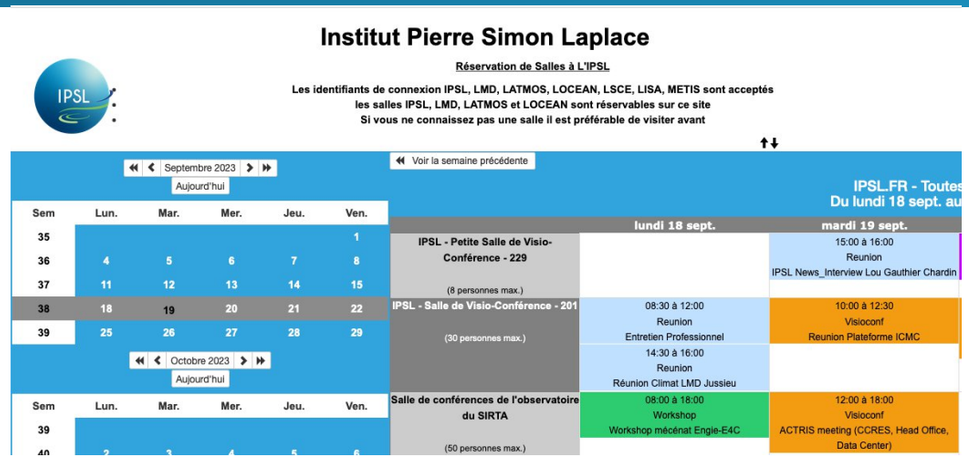
- Syntax

```
ssh -N -L <local port>:<intranet host FQDN>:<forwarded port> [username@]<jump host FQDN>
```



IPSL rooms reservation without VPN

IPSL room reservation page (https://resa.ipsl.fr) is only accessible within the IPSL networks, not from Internet. Machines outside of IPSL networks can access to this page though IPSL VPN or by a SSH local port forwarding:



- From localhost in Internet:

```
user@port-500:~$ ssh -N -L 8000:resa.ipsl.fr:443 user@ssh-gw.ipsl.fr
```

- Then in a local web browser, open https://localhost:8000 and accept SSL exception.

- The config file can be configured like this:

```
Host resa
    HostName ssh-gw.ipsl.fr
    User user
    IdentityFile ~/.ssh/ipsl_id_ed25519
    LocalForward 8000 resa.ipsl.fr:443 # There is not ":" after "8000" and before "resa"
    RequestTTY no # Prevent login into ssh-gw.ipsl.fr
```

So as to replace the previous command line by: `ssh resa`

JupyterLab as cluster job 1/2

doc

- Start JupyterLab instance on spirit cluster interactive job with configured connection (~/.ssh/config)

```
user@port-500:~$ ssh spirit1
user@spirit1:~$ srun --pty --time 2:00:00 bash # Can specify memory and cpu cores here.
user@spirit64-01:~$ module load pangeo-meso/2023.04.15 # Load any module where jupyter is
provided!
(pangeo-meso-2023.04.15) user@spirit64-01:~$ jupyter lab --no-browser --ip=0.0.0.0
--port=8888 # <number between 10000 and 15000>
...
[I 2023-11-03 10:35:25.955 ServerApp]
http://127.0.0.1:8888/lab?token=cf4455a95cb689e69717515b38b6b554b51cbd80a7f66ec2
[I 2023-11-03 10:35:25.955 ServerApp] Use Control-C to stop this server and shut down all
kernels (twice to skip confirmation).
[C 2023-11-03 10:35:25.959 ServerApp]

    To access the server, open this file in a browser:
        file:///home/user/.local/share/jupyter/runtime/jpserver-257415-open.html
    Or copy and paste one of these URLs:
        http://spirit64-01:8888/lab?token=cf4455a95cb689e69717515b38b6b554b51cbd80a7f66ec2
        http://127.0.0.1:8888/lab?token=cf4455a95cb689e69717515b38b6b554b51cbd80a7f66ec2
```

JupyterLab as cluster job 2/2

doc

- Setup SSH local port forwarding with configured connection (~/.ssh/config)

In another local shell (not on spirit cluster!):

```
ssh -N -L 8888:spirit64-01:8888 spirit1 # This command line does not return anything (-L)!
```

- Display JupyterLab

Just copy/past the previous URL containing 127.0.0.1 (localhost) into a local Web browser.

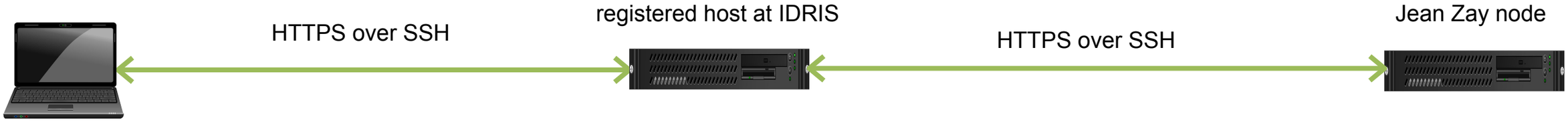
- At the end

1. Shutdown Jupyterlab (file menu then shutdown).
2. Issue CTRL + c shortcut in the shell where ssh local port forwarding was executed.
3. Exit interactive job.

JupyterHub at Jean-Zay

doc

Accessing to JupyterHub via SOCKS proxy:

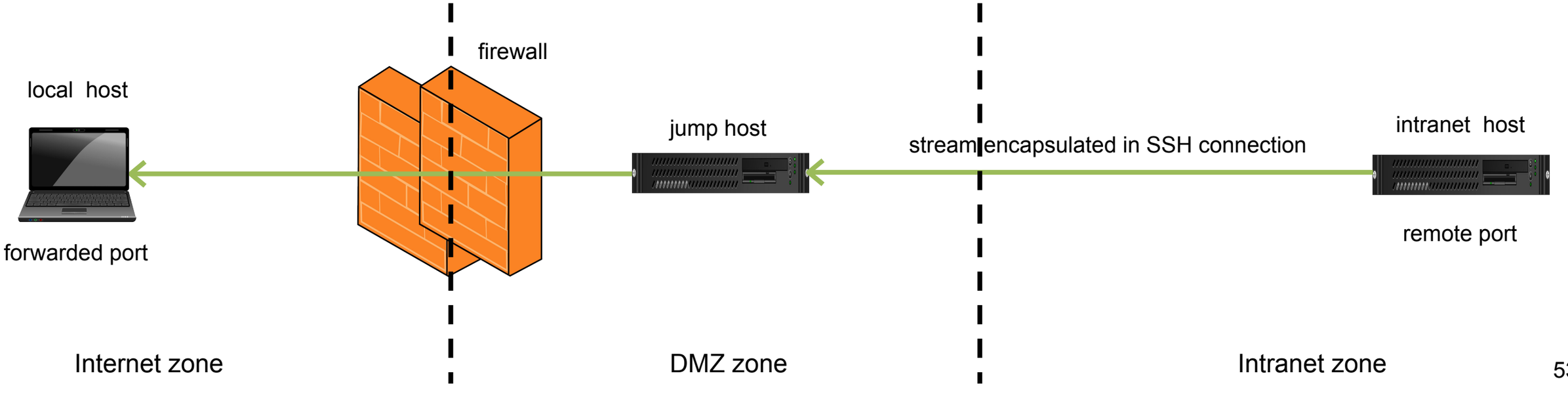


Remote port forwarding

- Principle
  - Relay a port on an intranet host to an Internet host, through a jump host.
  - Any service that opens the forwarded remote port on Internet host, is accessible on the intranet host. In other words, intranet host is able to access to services that run on Internet host. Use cases are fairly rare.

- Syntax

```
ssh -N -R <remote port>:<intranet host FQDN>:<forwarded port> [username@]<jump host FQDN>
```



MERCI - THANK YOU

Institut Pierre Simon Laplace IPSL