Using ssh keys

Basic description on how to use ssh keys

- Create a ssh key
- Security
- Configuring SSH
- Using SSH tunnels
- Enable access to a remote host

Create a ssh key

Access to the INCD computing clusters is performed via SSH and requires the use of SSH keys for authentication. Authentication with passwords is not supported. Each SSH key pair has two components a **public key** that must be added to the hosts to be remotely accessed, and a **private key** that must remain in the user workstation or laptop machine. The private key must be protected with a **strong password**. The users must generate their own SSH key pair in a machine of their own (workstation, laptop, etc). To generate your SSH key pair follow these instructions.

Linux and macOS

- Users must generate the SSH key pair in a computer of their own (desktop, notebook etc).
- The passphrase is used to protect the private key, very IMPORTANT please choose a strong password with uppercase and lowercase characters, numbers and symbols.

NOTE: xxx will be rsa, ed25519 or something similar. And you must check and replace it accordling in the instructions below.

```
$ ssh-keygen -b 4096
Generating public/private xxx key pair.
Enter file in which to save the key (/home/username/.ssh/id_xxx):
Created directory '/home/username/.ssh'.
Enter passphrase (empty for no passphrase):          ----> IMPORTANT: Choose a strong password
Enter same passphrase again:                         ----> IMPORTANT: Choose a strong password
Your identification has been saved in /home/username/.ssh/id_xxx
Your public key has been saved in /home/username/.ssh/id_xxx.pub
```

- ssh-keygen will create a pair of keys, private (id_xxx) and public (id_xxx.pub), these files are created in the user home directory usually under $HOME/.ssh
- The file and directory protections of $HOME/.ssh should be as follows:

```
ls -la $HOME/.ssh/
total 8
drwx------ 3 username group 4096 Jan 11 18:12 .
-rw------- 1 username group 1743 Feb 19 10:52 id_xxx
-rw-r--r-- 1 username group  404 Feb 19 10:52 id_xxx.pub
```

- Correct File permissions using chmod

```
chmod 700 .ssh
chmod 644 id_xxx.pub
chmod 600 id_xxx
```

- Users must send to the INCD administrators **only** the public key **id_xxx.pub** (full content of the file).
- The private key must must be kept private and must NEVER be shared with other persons.

Microsoft Windows

- For Windows users accessing the INCD public machines we recommend the use of **terminal emulators** like **MobaXterm**.
  - MobaXterm works both as a terminal and X windows server, this allows to display in your desktop the graphical X11 windows from the remote Linux host.
  - MobaXterm supports file transfer via the embbed SFTP browser.
  - MobaXterm documentation is available **here**
  - Key pairs can be generated with `ssh-keygen -b 4096`
  - You can also create and manage your SSH keys using the embedded **MobaKeyGen** application (available from the "Tools" menu).

- Windows users can also generate ssh-keys using **Putty**
  - Download and install Putty
  - Generate the key in your Windows machine see these examples:
    - Youtube
    - HowTo
  - IMPORTANT: notice that Putty is only a text terminal and does not work as X windows graphics server and does not support file transfer.

- Users must send to the INCD administrators **only** the public key **id_rsa.pub**
- The private key must must be kept private and must NEVER be shared with other persons.

Login does not work

- If the INCD helpdesk confirms that your public key has been installed and still you cannot login please check the following:
  1. That you are trying to access the correct INCD login hostname as indicated by the INCD helpdesk.
  2. That you are trying to access the INCD host from the same machine and user account where you generated the SSH key pair.
  3. That the permissions and ownership of your SSH directory and contained files are correct. The relevant Linux directory and files are below, see if they match the protections described in the Linux section above:
     - $HOME/.ssh
     - $HOME/.ssh/id_rsa
     - $HOME/.ssh/id_rsa.pub
  4. That the SSH private key password is correct. In Linux you can do this by trying to load the private key into the SSH agent with the command: `ssh-add`
  5. That the INCD login host is reachable from your machine. From Linux you can use the command `nmap -P0 -p22 hostname` the returned port STATE for the SERVICE *ssh* (PORT 22/TCP) must be **"open"**. If the hostname does not resolve or the STATE is different from **"open"** (e.g. filtered) you may have a network connectivity problem.
  6. If you are on Windows consider to install a Linux virtual machine and perform the SSH access from that Linux VM. Notice that in this case you need to place the keypair (id_rsa and id_rsa.pub) in the Linux virtual machine or generate a new key pair in Linux and send it to the INCD administrators. To install Linux on windows you can use:
     - An hypervisor such as **VirtualBox**
     - The **Windows Subsystem for Linux (WSL)**
- If you still can't login please contact the INCD helpdesk and provide details on the error and verification steps that you already performed.

Security

- Choose strong passwords - at least 9 characters long, a mixture of alphanumeric mixed case and symbol characters. The password should be completely different from the password you use on any other system.
- **Never use the same password across different systems !**
- **NEVER copy your SSH private key to systems that you do not control!** The private key should remain in your .ssh directory on the system you generated it and should be readable only by you. If you need to login from two systems such as a laptop and a workstation you can copy the key pair to both systems ONLY if you really trust both. When copying always check that the copied files are only readable by yourself.
- SSH key passphrases must be as secure as other passwords.
- **Never setup passphraseless** ssh keys to allow unauthenticated login access between systems !!!

WARNING: Incorrectly configuring SSH keys can leave your accounts vulnerable to attack and, more importantly, can provide attackers with a trivial means to access remote systems with potential legal consequences for yourself. It is your responsability to keep your SSH authentication and your user account secure.

Configuring SSH

In order to simplify SSH access to remote hosts we recommend INCD users to adopt the following recommendatios and configuration defaults.

Enabling SSH agents and X11 forwarding

Activate the forwarding of both SSH authentication credentials and X11 graphical windows. This will facilitate your SSH access by enabling:

- Logging in across hosts without having to enter passwords or other credentials (*ForwardAgent*);
- X11 applications to forward their GUI back to your workstation display using the SSH connection (*ForwardX11* and *ForwardX11Trusted*).

Notice that these are two independent features unrelated to each other, you can activate one or both. If you do not require the forwarding of graphical X11 windows you can skip it. Similarly if you do not plan to access other hosts behind the login host or if you do not trust the remote host you should skip the forwarding of SSH credentials (*ForwardAgent*).

To activate both options the following configuration steps should be performed in the local workstation (PC or laptop desktop) from which the remote INCD hosts will be accessed.

- Edit the local SSH config file, either the system configuration file or the user specific configuration file in your home directory.

  ```
  $sudo vi /etc/ssh/ssh_config
  ```

  or

  ```
  $sudo vi $HOME/.ssh/config
  ```

- Add the following options:

  ```
  Host *
    ForwardAgent yes
    ForwardX11 yes
  ```

```
ForwardX11Trusted yes
```

- **Disclaimer:** In some operating systems the location of the SSH configuration file may change please check your OS for details.

Alternatively the same SSH forwarding options can be activated from the command line for each connection by invoking SSH with the corresponding flags **-A -X** and **-Y**:

```
ssh -A -X -Y  remote-hostname
```

More information about SSH and port forwarding can be found in:

- The INCD wiki page on SSH port forwarding available **here**
- The SSH Forwarding guide

Using SSH tunnels

In some cases you may need to access remote hosts that have private IP addresses or are protected behind a firewall. These hosts might be only accessible through an intermediate login host. The SSH tunnels (SSH port forwarding) facilitate access to multiple hosts protected behind a single public host exposed to the Internet. Once the tunnels are established the hosts behind the firewall can be directly accessed from your machine. Examples where tunnels can be beneficial:

- When using the INCD cloud service you may have multiple VMs with private IP addresses acessible through a single front-end VM having a single public IP address. This enables you to access remote VMs with private addresses and save scarse IPv4 address space.
- Some INCD services are not directly exposed to the Internet and may require passing through an intermediate login host for added security.

In the following examples the hostnames *public.a.acnca.pt* and *internal.a.acnca.pt* as well as the port numbers are placeholders that should be replaced by actual real hostnames and port numbers according to your scenario.

jump over SSH hosts easily from the command line

- If you just need to connect via SSH by jumping over a reachable host you can use the jump host functionality of SSH whose syntax is described below and where:
  - **internal-remote-host** is the host or IP address behind the firewall that you want to reach via SSH.
  - **public-remote-host** the remote host or IP address of the intermediate host that is exposed to the Internet through which you will connect to the **internal-remote-host**.
  - The username is optional and can be different for the public and internal host.

Basic jump host syntax:

```
ssh -J username@public-remote-host   username@internal-remote-host
```

Example of using the SSH jump host functionality to access the SSH port (port number 22) of the remote host *internal.a.acnca.pt* through a publicly reachable host *public.a.acnca.pt*. Most of the examples in this page are valid both for ssh and its related commands such as scp and sftp.

```
ssh -J public.a.acnca.pt  username@internal.a.acnca.pt
scp -J public.a.acnca.pt  mylocalfile  username@internal.a.acnca.pt:
sftp -J public.a.acnca.pt username@internal.a.acnca.pt


ssh -J username@public.a.acnca.pt  username@internal.a.acnca.pt
```

forwarding of X11 windows

- SSH enables the forwarding of X windows.
- This requires that your local machine has an X11 display server. If you are on Windows you can use **MobaXterm** that works both as ssh client and X server.
- For more information on this topic see **here**

port forwarding from local to remote in detail

- Port forwarding enables multiple tunnels over a single SSH connection thus exposing multiple remote hosts and ports to be directly accessed by SSH-related and SSH-unrelated applications (such as forwarding HTTP/HTTPS ports over SSH).
- The port forwarding is usually defined upon the connection to the intermediate host that is directly acessible. An SSH session to the intermediate host can simultaneously define one or more port forwardings using the **-L** option.
- The basic syntax for SSH port forwarding is described below where:
  - **local-port** is a TCP port of your local machine (your desktop, laptop, etc). Just pick one port that is not being used. This port should be above the privileged port range (above 1024) and preferrably outside of the IP ephemeral port range to avoid collisions with dynamically allocated ports (for many Linux systems select a port outside of the range 32768-60999).
  - **internal-remote-host** is an IP address or hostname of a remote host behind the firewall.
  - **internal-remote-port** is the TCP port of the internal-remote-host that you want to access.
  - **public-remote-host** the remote host or IP address of the intermediate host that is exposed to the Internet.

Basic port forwarding syntax:

```
ssh -L local-port:internal-remote-host:internal-remote-port   public-remote-host
```

Example of establishing a tunnel to access the SSH port (port number 22) of the remote host *internal.a.acnca.pt* through a publicly reachable host *public.a.acnca.pt*. We pick a random *local-port* (e.g. 31732) that is then mapped to the port 22 of *internal.a.acnca.pt*.

```
ssh -L 31732:internal.a.acnca.pt:22   public.a.acnca.pt
```

Once the SSH connection to *public.a.acnca.pt* is established the host *internal.a.acnca.pt* can be directly accessed from your local host by connecting to the *local-port* on the loopback address. Just create a second command line terminal and try:

```
ssh -p 31732 username@127.0.0.1
scp -P 31732 mylocalfile  username@127.0.0.1:
sftp -P 31732 127.0.0.1
```

Once the SSH connection to the intermediate host is closed the tunnels will stop working. Therefore the initial SSH connection must be kept alive while needed and the connections to the remote internals hosts must be performed from a separate local terminal window.

Another example where both port 22 (SSH) and port 80 (HTTP) are both mapped to local ports. With this both ports are forwarded over a single SSH connection. The access to the web server from the local host can be tested using curl.

```
ssh -L 31732:internal.a.acnca.pt:22 -L 8080:internal.a.acnca.pt:80  public.a.acnca.pt


curl http://127.0.0.1:8080
```

port forwarding from remote to local in detail

- This enables a remote host behind the firewall to access services in your local network. Basically this enables port forwarding in the reverse direction, from the remote network to the local network.
- This can be potentially dangerous for you as it enables users in the remote machine or remote network to access your local machines.
- The basic syntax for port forwarding from the remote side to the local side is described below where:

- **remote-port-on-public-host** is a port number from the *public-remote-host* Just pick one port that is not being used above the privileged port range (above 1024) and preferrably outside of the IP ephemeral port range to avoid collisions with dynamically allocated ports (for many Linux systems outside the range of 32768-60999).
- **local-host** a host in the local network of the user to be accessed from the *public-remote-host*.
- **local-port** the port number of the *localhost* to be accessed from the *public-remote-host*.
- **public-remote-host** the remote host or IP address of the intermediate host that is exposed to the Internet.

Basic remote port forwarding syntax:

```
ssh -R remote-port-on-public-host:local-host:local-port  public-remote-host
```

Example of forwarding the *remote-port* number 65532 on *public.a.acnca.pt* to the port 22 of the local host named *myotherlocalhost*.

```
ssh -R 65532:myotherlocalhost:22  public.a.acnca.pt
```

Then from *public.a.acnca.pt* you can access myotherlocalhost in your local network with:

```
ssh -p 65532 127.0.0.1
```

jump host via config file

- Configuring SSH via jump host in your $HOME/.ssh/config enables the configuration to be stored. The configuration syntax is below where:
  - **Host** is a name choosen by you to identify this mapping, this name may be different from the hostname and will be recognized as a hostname only by SSH.
  - **Hostname** the real hostname of the internal host that you want to access behind the firewall.
  - **User** the username for the internal host identified by *Hostname*.
  - **ProxyJump** the remote host or IP address of the intermediate host that is exposed to the Internet.

The basic configuration for remote access via a jump host

```
Host name-for-the-internal-remote-host
Hostname actual-internal-remote-host
User username-in-actual-internal-remote-host
```

```
ProxyJump username@public-remote-host
HostKeyAlias name-for-the-internal-remote-host
```

Example of configuration:

```
Host ncg-internal
Hostname internal.a.acnca.pt
User username-for-internal
ProxyJump username@public.a.acnca.pt
HostKeyAlias ncg-internal
```

Accessing the configured host:

```
ssh ncg-internal
```

port forwarding via config file

- Configuring port forwarding in your $HOME/.ssh/config requires configuring two hosts. The first will be the intermediate host accessible through the Internet and the second the internal host. The fields are as follows:
  - **public-remote-host** the remote host or IP address of the intermediate host that is exposed to the Internet.
  - **LocalForward** the mapping of a *local-port* to a remote internal hostname or IP address and port number that are behind the firewall.
  - **name-for-the-internal-remote-host** is a name choosen by you to identify this mapping, this name may be different from the actual hostname and will be recognized as a hostname only by SSH.
  - **local-port** must be the same across the two host definitions and is a local TCP port number that will be mapped to the *internal-remote-host* and *internal-remote-port*.

The basic configuration for port forwarding:

```
Host public-remote-host
LogLevel FATAL
LocalForward local-port internal-remote-host:internal-remote-port


Host name-for-the-internal-remote-host
```

```
Hostname 127.0.0.1
User username
Port local-port
HostKeyAlias name-for-the-internal-remote-host
```

Example of port forwarding for both SSH (port 22) and HTTP (port 80):

```
Host public.a.acnca.pt
LogLevel FATAL
LocalForward 31732 internal.a.acnca.pt:22
LocalForward 8080  internal.a.acnca.pt:80

Host ncg-internal
Hostname 127.0.0.1
User username
Port 31732
HostKeyAlias ncg-internal
```

Accessing the configured host

```
ssh ncg-internal
curl http://127.0.0.1:8080
```

(*) notice that you cannot use the hostname *ncg-internal* with curl as this is a hostname that is only recognized by SSH related applications (ssh, scp, sftp).

using socks proxies

- SSH supports socks4 and socks5 proxying. This allows socks enabled applications to access hosts behind the firewall. This is only valid for socks enabled applications e.g. firefox.
- The syntax is below where:
  - **public-remote-host** the remote host or IP address of the intermediate host that is exposed to the Internet
  - **local-port** as in the other cases is a TCP port of your local machine (your desktop, laptop, etc). Just pick one port that is not being used. This port should be above the privileged port range (above 1024) and preferrably outside of the IP ephemeral port range to avoid collisions with dynamically allocated ports (for many Linux systems select a port outside of the range 32768-60999).

The basic command line usage to establish a socks server via SSH:

```
ssh -D local-port  username@public-remote-host
```

The basic $HOME/.ssh/config setup to establish a socks server:

```
Host public-remote-host
DynamicForward local-port
User username
```

Example of socks proxy use with Firefox:

```
ssh -D 31733 username@public.a.acnca.pt
```

Then in firefox:

1. Goto *Preferences->Network Settings*:
2. Select *Manual proxy configuration*
3. Enter in *SOCKS Host*: 127.0.0.1
4. Enter in *Port*: 31733
5. If needed add local networks to be excluded from the proxy using *No proxy for*
6. If needed add the options related to DNS proxying

(*) notice that all your http request will now be forwarded through the socks proxy, to restore the previous settings choose *No proxy* in the *Preferences->Network Settings*.

(*) as in the other examples the TCP port number to be allocated might need to be adjusted to match a free port number in your local machine (desktop, laptop, etc).

using sshuttle

The tool sshuttle is available in many Linux distributions. It uses SSH in combination with iptables and other functionalities to easily establish tunnels with better performance and a behavior more similar to a VPN. Instead of establishing a direct end-to-end TCP connection to the remote internal hosts, sshutle intercepts local packets, sends their payload over SSH and establishes connections from the remote public host to the internal hosts. This prevents some of the performance issues associated with TCP end-to-end connections through SSH inner tunnels and also enables other capabilities such as accessing more transparently hosts and services behind the firewall.

Requirements:

- Python 2.3 or higher on the remote server side.
- sshuttle requires root privileges in your local machine (desktop, laptop, etc), root privileges in the remote machine are not required.

- sshuttle must be installed in the local client side, sshuttle is not required on the remote host.
- Create an sshutle configuration file for instance in $HOME/.ssh/sshutle-ncg.conf with the following content:
  - **remote networks plus prefixes** one per line (e.g. 192.168.1.0/24)
  - **--auto-nets** optional to add networks know by the remote host side.
  - **--dns** optionally enables the capture of dns requests and forwards them to the remote side, use carefully as it may disrupt your local DNS queries. Also if you have multiple sshuttle instances you can only have one of them capturing and forwarding the DNS queries.
  - **--auto-hosts** to enable discovery of hosts in the remote side
  - **--seed-hosts** followed by a line containing remote hostnames separated by commas that you want to use
  - **--remote** followed by a line with the hostname of the public remote host to act as intermediate host

Template for the sshutle configuration file:

```
ip-network-A/ip-prefix-A
ip-network-B/ip-prefix-B
ip-network-X/ip-prefix-X
--auto-nets
--auto-hosts
--seed-hosts
internal-remote-host-01,internal-remote-host-02, ...
--remote
username@public-remote-host
```

Example that needs to be adjusted to your actual needs:

```
192.168.1.0/24
192.168.2.0/24
--auto-hosts
--seed-hosts
internal.a.acnca.pt, otherinternal.a.acnca.pt, anotherinternal.a.acnca.pt
--remote
username@public.a.acnca.pt
```

Invoking sshutle from the command line:

```
sshuttle @$HOME/.ssh/sshutle-ncg.conf
```

Then you can ssh into the hosts directly:

```
ssh internal.a.acnca.pt
```

If the remote internal host has other services running such as a web server they will be also accessible:

```
curl http://otherinternal.a.acnca.pt
```

Enable access to a remote host

In most cases you will not need to add your own SSH public key to other INCD remote hosts as there are other processes to do so, namelly:

- To get access to the INCD login hosts you will be required to send your SSH public key to the INCD administrators that will install it where needed and will report back to you.
- When using the INCD cloud services you may need to add your SSH key to a VM managed by yourself. Usually this is done through the Openstack command line interface or through the Openstack Horizon dashboard when the machine is first created and does not require other manual intervention. See the INCD cloud documentation **here**.

- However if you need to add a key manually to a remote host account the required steps are:
  1. login into the remote host account or access its home directory through the root account
  2. check if the account directory `~user/.ssh` exists in the home directory of the remote user, if not create it with `mkdir ~user/.ssh; chmod u=rwx ~user/.ssh`
  3. check if the file `~user/.ssh/authorized_keys` exists under `~user/.ssh`, if not create the file with `touch ~user/.ssh/authorized_keys; chmod u=rw ~user/.ssh/authorized_keys`
  4. copy the public key to the remote host and append it to the file `~user/.ssh/authorized_keys` make sure not to overwrite other keys that may already exist.
  5. If you did the above steps from the root account make sure the files ownership are correct and if not change them to the correct uid and gid with `chown -R user.group ~user/.ssh`