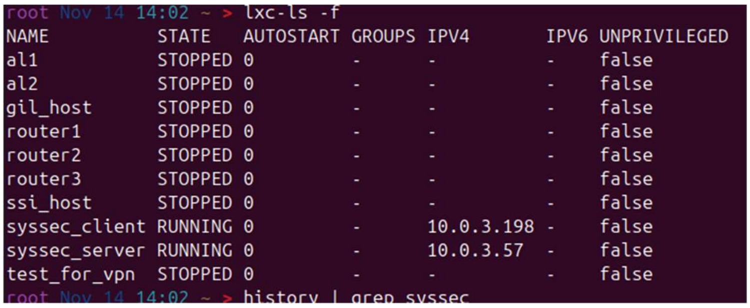
TP1: The SSH protocol and its openssh implementation

Ifrim Vasile-Alexandru
M2SSI

1. Network setup

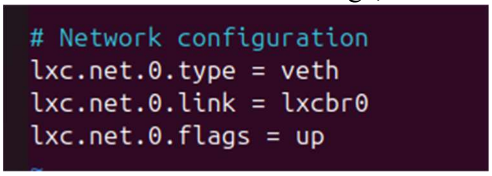
To setup a simple network, we create 2 lxc containers named syssec_client and syssec_server.

```
$ lxc-create --name syssec_client --template download -- --dist ubuntu --release jammy --arch amd64
$ lxc-create --name syssec_client --template download -- --dist ubuntu --release jammy --arch amd64
```



```
root Nov 14 14:02 ~ > lxc-ls -f
NAME          STATE   AUTOSTART GROUPS IPV4       IPV6 UNPRIVILEGED
al1           STOPPED 0         -      -          -    false
al2           STOPPED 0         -      -          -    false
gil_host      STOPPED 0         -      -          -    false
router1       STOPPED 0         -      -          -    false
router2       STOPPED 0         -      -          -    false
router3       STOPPED 0         -      -          -    false
ssi_host      STOPPED 0         -      -          -    false
syssec_client RUNNING 0         -      10.0.3.198 -    false
syssec_server RUNNING 0         -      10.0.3.57  -    false
test_for_vpn  STOPPED 0         -      -          -    false
root Nov 14 14:02 ~ > history | grep syssec
```

The 2 containers will be connected to the default lxc bridge, lxcbr0.



```
# Network configuration
lxc.net.0.type = veth
lxc.net.0.link = lxcbr0
lxc.net.0.flags = up
```

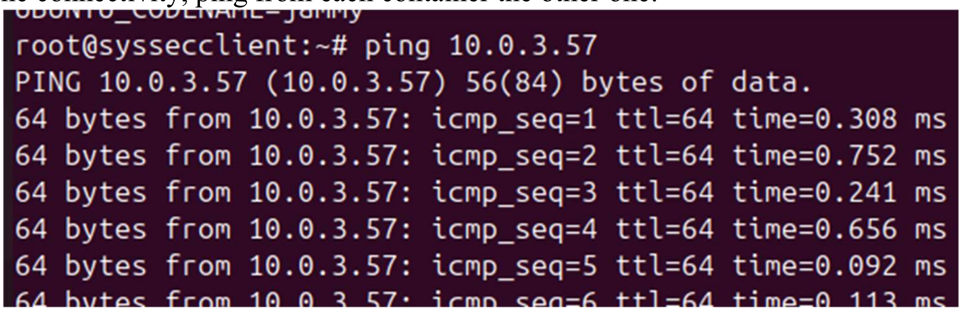
/var/lib/lxc/syssec_client/config file; similar for syssec_server

To see containers' OS info,

```
$ cat /etc/os-release
```

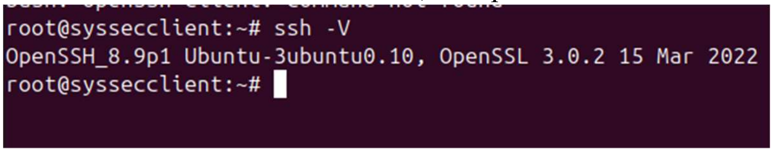
```
root@sysecclient:~# cat /etc/os-release
PRETTY_NAME="Ubuntu 22.04.5 LTS"
NAME="Ubuntu"
VERSION_ID="22.04"
VERSION="22.04.5 LTS (Jammy Jellyfish)"
VERSION_CODENAME=jammy
ID=ubuntu
ID_LIKE=debian
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
UBUNTU_CODENAME=jammy
root@sysecclient:~#
```

To check the connectivity, ping from each container the other one:

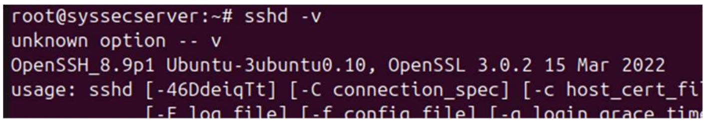


```
root@sysecclient:~# ping 10.0.3.57
PING 10.0.3.57 (10.0.3.57) 56(84) bytes of data.
64 bytes from 10.0.3.57: icmp_seq=1 ttl=64 time=0.308 ms
64 bytes from 10.0.3.57: icmp_seq=2 ttl=64 time=0.752 ms
64 bytes from 10.0.3.57: icmp_seq=3 ttl=64 time=0.241 ms
64 bytes from 10.0.3.57: icmp_seq=4 ttl=64 time=0.656 ms
64 bytes from 10.0.3.57: icmp_seq=5 ttl=64 time=0.092 ms
64 bytes from 10.0.3.57: icmp_seq=6 ttl=64 time=0.113 ms
```

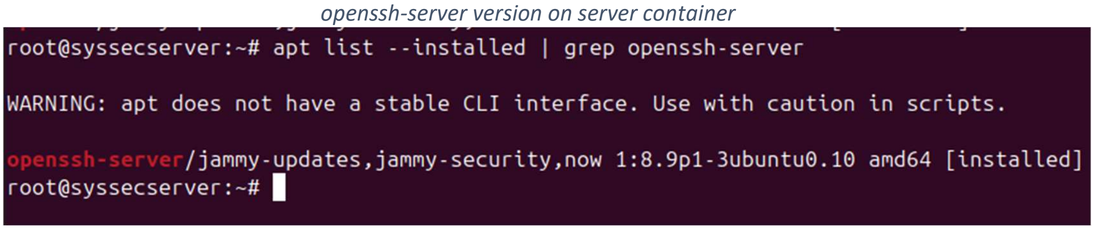
And to check openssh-client version we use `ssh -V`, for openssh-server we use `sshd -V`



openssh-client version on client container

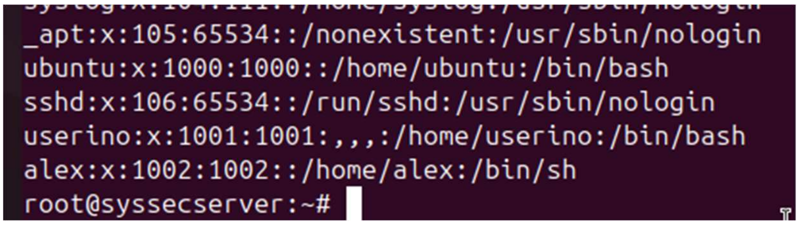


openssh-server version on server container



Finally, to create a new user on each container, and then add a password to that account

```
$ useradd $username
$ passwd $username
```



`cat /etc/passwd` to see all system users

2. Telnet

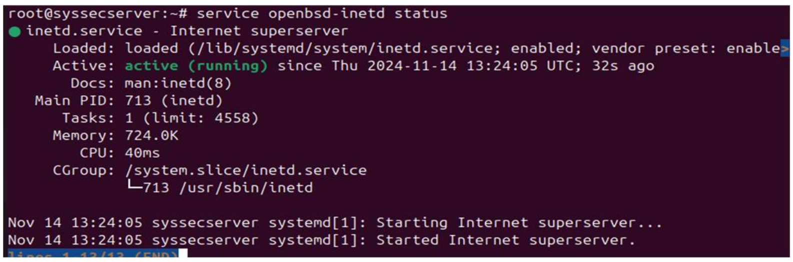
To install a telnet server and "start" it,

```
$ apt install telnetd
$ service openbsd-inetd start
```

`openbsd-inetd` is the OpenBSD implementation of the `inetd` super-server daemon (also known as a service dispatcher), providing internet services on each configured port (inetd listens for service requests specified in the `inetd.conf` file at a port defined in the services file); requests are served by spawning a process which runs the appropriate executable. So, `telnetd` is started `when inetd` receives a service request to connect to the TELNET port.

To check that the service has started successfully,

```
$ service openbsd-inetd status
```



On the client, we install the telnet client and connect to the server by:

```
$ apt install telnet
$ telnet $server_ip
```

We will be prompted for a user and password.

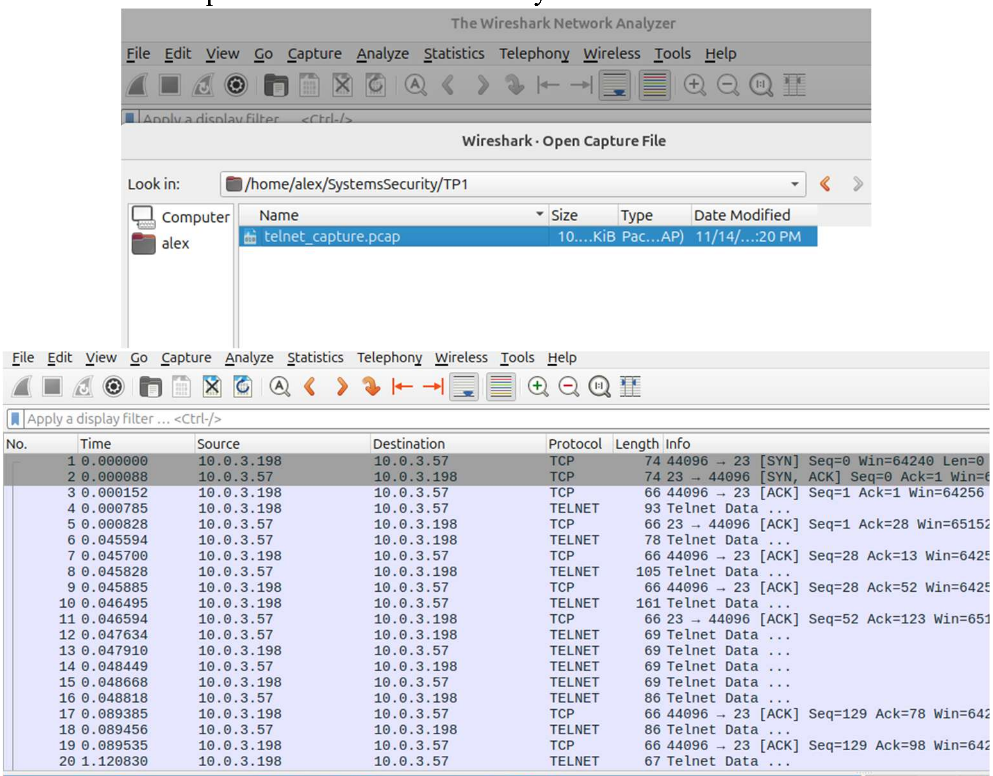
```
root@syssecclient:~# telnet 10.0.3.57
Trying 10.0.3.57...
Connected to 10.0.3.57.
Escape character is '^]'.
Ubuntu 22.04.5 LTS
syssecserver login: alex
Password:
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 6.8.0-48-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro
```

Next, we will start capturing, from the host, traffic going through the lxcbr0 bridge interface, filtering by TCP protocol and port 23. After, we will telnet from the client again. After logging in, we can close the connection, then the tcpdump process by CTRL-C, writing to telnet_capture.pcap.

```
$ sudo tcpdump -i lxcbr0 tcp port 23 -w telnet_capture.pcap
```

The capture file will be opened in Wireshark for analysis.



Going through the captured packets we will see that the login name, password and any other input from the client is sent in plaintext, one letter by packet.

Finally, to completely remove telnet we must delete telnetd and inetd.

```
$ service openbsd-inetd stop
$ apt remove --purge -y telnetd openbsd-inetd
```

3. SSH Server

To start the ssh service on the server and check its status,

```
$ service ssh start
$ service ssh status
```

```
root@syssecserver:~# service ssh status
● ssh.service - OpenBSD Secure Shell server
     Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: ena
     Active: active (running) since Thu 2024-11-14 13:02:15 UTC; 46min ago
       Docs: man:sshd(8)
             man:sshd_config(5)
   Main PID: 166 (sshd)
      Tasks: 1 (limit: 4558)
     Memory: 6.4M
        CPU: 161ms
     CGroup: /system.slice/ssh.service
             └─166 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups"

Nov 14 13:02:15 syssecserver systemd[1]: Starting OpenBSD Secure Shell server...
Nov 14 13:02:15 syssecserver sshd[166]: Server listening on 0.0.0.0 port 22.
Nov 14 13:02:15 syssecserver sshd[166]: Server listening on :: port 22.
Nov 14 13:02:15 syssecserver systemd[1]: Started OpenBSD Secure Shell server.
```

We can see from its status that the process is listening on port 22, but to confirm we can check that with the sockets util `ss`:

```
$ ss -atlp
$ # -a for all, -t for TCP, -l for listen, and -p for print the
$ # corresponding process
$ # if possible, ports will be printed as their IANA-assigned protocol
names
```

```
root@syssecserver:~# ss -atlp
State      Recv-Q     Send-Q         Local Address:Port          Peer Address:Port
Process
LISTEN     0          4096           127.0.0.53%lo:domain             0.0.0.0:*
 users:(("systemd-resolve",pid=152,fd=14))
LISTEN     0          128                  0.0.0.0:ssh                0.0.0.0:*
 users:(("sshd",pid=166,fd=3))
LISTEN     0          128                     [::]:ssh                   [::]:*
 users:(("sshd",pid=166,fd=4))
root@syssecserver:~#
```

The sshd keys are located in /etc/ssh:

```
root@syssecserver:~# ls /etc/ssh/ -l
total 540
-rw-r--r-- 1 root root 505426 Jun 26 13:11 moduli
-rw-r--r-- 1 root root   1650 Jun 26 13:11 ssh_config
drwxr-xr-x 2 root root   4096 Jun 26 13:11 ssh_config.d
-rw-r--r-- 1 root root   3254 Jun 26 13:11 sshd_config
drwxr-xr-x 2 root root   4096 Jun 26 13:11 sshd_config.d
-rw------- 1 root root    513 Nov 12 17:42 ssh_host_ecdsa_key
-rw-r--r-- 1 root root    179 Nov 12 17:42 ssh_host_ecdsa_key.pub
-rw------- 1 root root    411 Nov 12 17:42 ssh_host_ed25519_key
-rw-r--r-- 1 root root     99 Nov 12 17:42 ssh_host_ed25519_key.pub
-rw------- 1 root root   2602 Nov 12 17:42 ssh_host_rsa_key
-rw-r--r-- 1 root root    571 Nov 12 17:42 ssh_host_rsa_key.pub
-rw-r--r-- 1 root root    342 Dec  7  2020 ssh_import_id
```

To check the public keys,

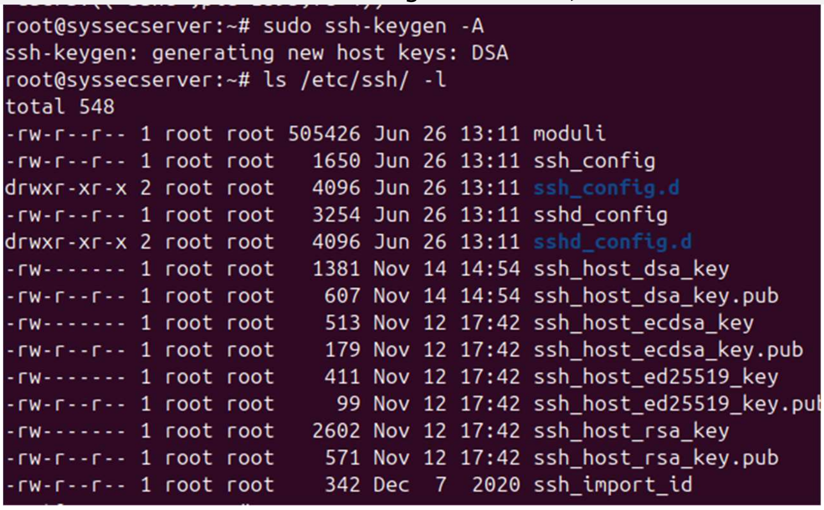
```
root@syssecserver:~# ssh-keygen -lf /etc/ssh/ssh_host_ecdsa_key.pub
256 SHA256:MynCevTg3OLjep9bjZjA5BflrFmPBPKChkh6Z8rafRQ root@syssecserver (ECDSA)
root@syssecserver:~# ssh-keygen -lf /etc/ssh/ssh_host_rsa_key.pub
3072 SHA256:5tTWXVbnJ5phjqtaXPC/89RepuHHJOq4NTWXI7saTTE root@syssecserver (RSA)
root@syssecserver:~#
```

ECDSA 256-bit key, RSA 3072-bit key

The fingerprint for a server's SSH keys should be unique. If another machine's fingerprint is identical, it indicates that one machine was cloned from the other, and/or the keys were accidentally copied.

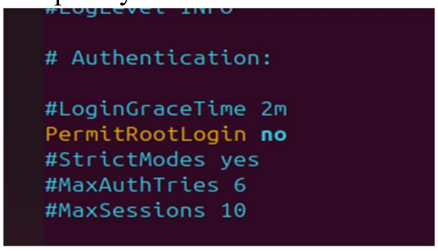
To regenerate keys,

```
$ sudo rm /etc/ssh/ssh_host_*
$ sudo ssh-keygen -A # generate all missing host keys
$ sudo service sshd restart # after generation, a restart is needed
```



Access to the private host keys should be permited only to their owner, only for reading and writing; any other user should be prevented from reading/modifying them.

In `/etc/ssh/sshd_config`, the setting for root login is commented initially; the default behavior depends from Linux distribution to distribution: it could allow root login with password, or only with SSH keys (password login is disabled). To prevent any ambiguity and reduce the risk of root abuse, root login must be disabled completely.



Setting `PrintLastLog` to `Yes` will display the date, time, and source IP of the last successful login when a user logs in; it helps in detecting unauthorized access, unexpected login times or IPs.

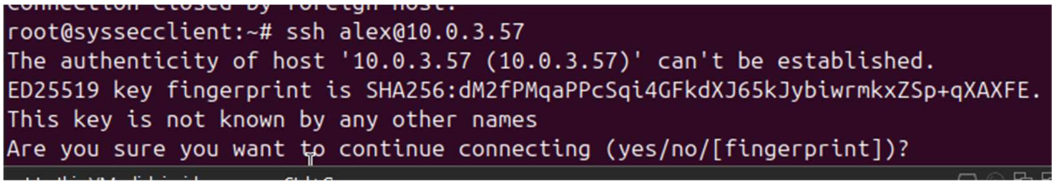
For any more configurations, we must make clear the differences between a few files:

- `/etc/ssh/sshd_config`: configures the ssh server (settings such as allowed authentication methods, login permissions, logging, and SSH port)
- `/etc/ssh/ssh_config`: configures the SSH client system-wide; define default settings for outgoing SSH connections, like preferred key types or timeout options
- `~/.ssh/config`: configures SSH client settings for the individual user; each user can override this way the ssh_config for their own connections

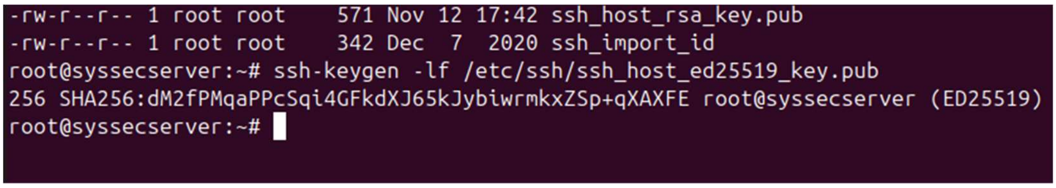
4. Client password authentication

The message "The authenticity of host '…' can't be established" means that the SSH client does not have the server's **host key** stored in its `~/.ssh/known_hosts`, where previously trusted servers are remembered. Since the client hasn't seen this key before, it can't confirm the authenticity of the server.

To check the host key's fingerprint, we compare it to the one directly on the server (`ssh-keygen -lf`) of the server's public key. If they match, the server's identity is correct. Then, to accept, we input ‘`yes`’ and the server’s key will be stored in `~/.ssh/known_hosts`. Future connections will be trusted.



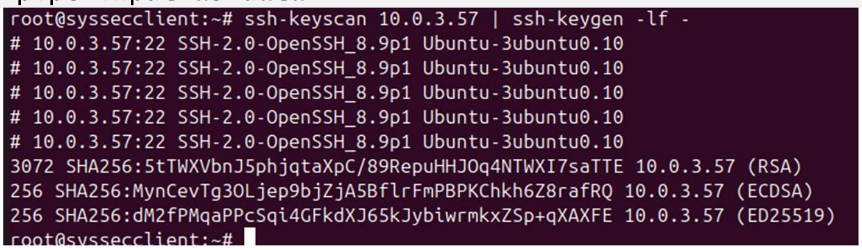
Explicit server authentication



Comparing the fingerprint of the pub key directly from the server to the one reported when attempting to connect shows that they are identical

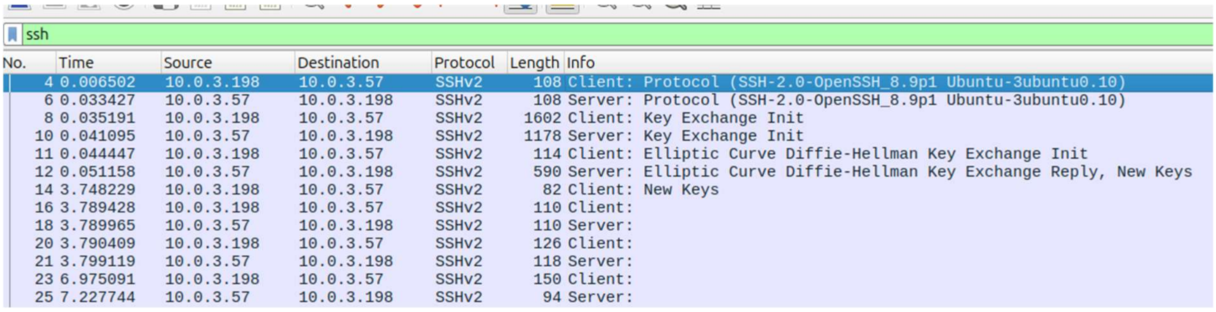
We can also scan from the client before attempting to authenticate to the server. This will list all the server’s public keys’ fingerprints.

```
   $ ssh-keyscan 10.0.3.57 | ssh-keygen -lf -
   $ # -l shows fingerprint of specified public key file
   $ # -f specifies the filename of the key file, by using ‘-‘ after we
redirect the pipe input as data
```

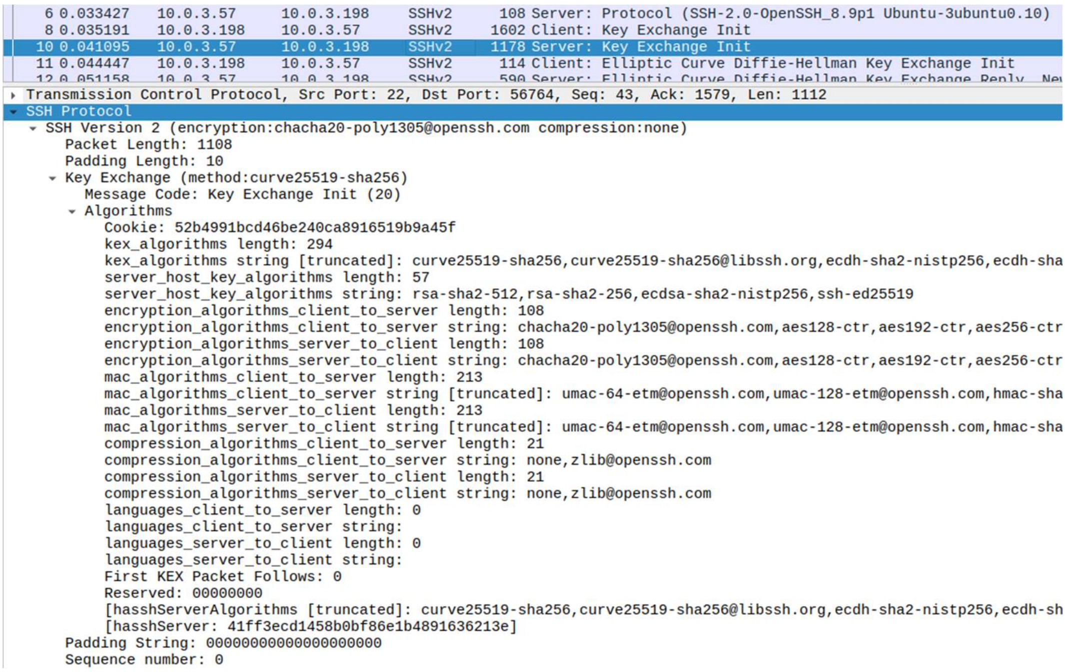


As described by RFC-4253, the SSH protocol is the basis in providing strong encryption, server authentication, and integrity protection; it may also provide compression. The first step in establishing a SSH connection is composed of:

- Identifying protocol versions used;
- Initiating the key exchange by agreeing on name-lists of supported encryption, MAC, and compression algorithms to use
    - for encryption, different ciphers in each direction **can** be used;
    - each packet’s MAC is computed from a shared secret, packet sequence number and the contents to protect data integrity; the MAC algorithms for each direction **must** be chosen independently and run independently, but in practice the same algorithm **can** be used
    - The Diffie-Hellman key exchange method specifies how one-time session keys are generated for encryption and for authentication, how the server authentication is done;
    - A ‘cookie’ **must** be a random value generated to make it impossible for either side to fully determine the keys and the session identifiers



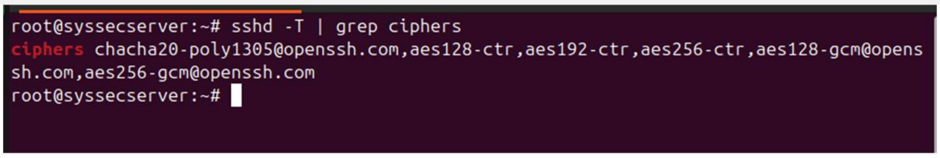
- The key exchange produces 2 values: a shared secret K and an exchange hash (additionally used as session identifier); encryption and authentication keys are derived from these
- An explicit server authentication is used if the key exchange messages include a signature or other proof of the server's authenticity; otherwise, implicit server authentication is used if, in order to prove it, the server also has to prove that it knows the shared secret, K.
- The key exchange ends with the 'New keys' message
- After it, the client requests a service (ex. Ssh-userauth, ssh-connection)



Key exchange begins by each side sending this algorithm negociation packet (as specified by RFC-4253, section 7.1)

To check what default symmetric algorithms are used by the server we can look into the configuration file by:

```
$ sshd -T | grep ciphers
$ # -T checks the validity of the configuration file, then outputs the effective conf to stdout
```



We can confirm by looking at the captured key exchange packet that indeed this set of ciphers was declared by the server in the `encryption_algorithms_server_to_client` key-value.

For allowed MACS,

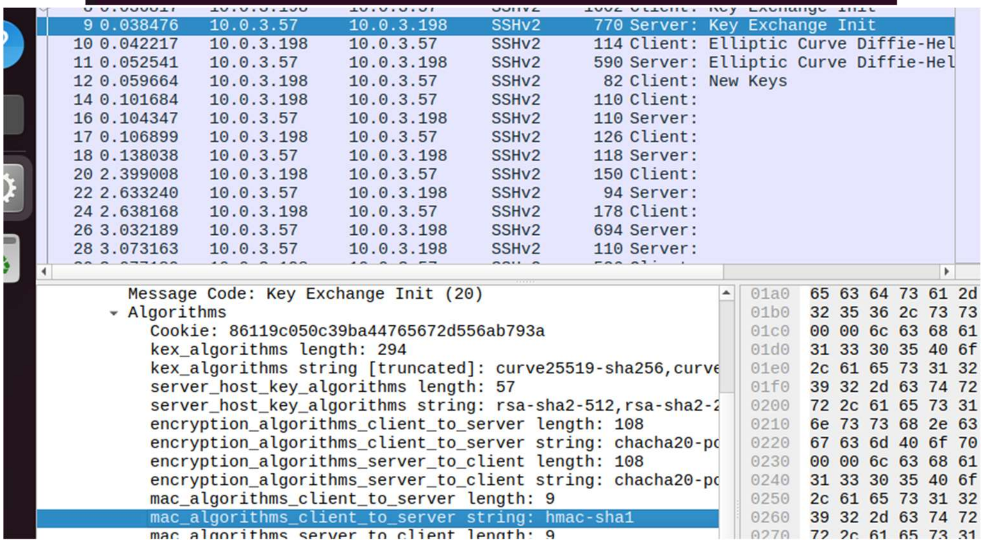
```
$ sshd -T | grep macs
```

```
root@syssecserver:~# sshd -T | grep macs
macs umac-64-etm@openssh.com,umac-128-etm@openssh.com,hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@openssh.com,hmac-sha1-etm@openssh.com,umac-64@openssh.com,umac-128@openssh.com,hmac-sha2-256,hmac-sha2-512,hmac-sha1
root@syssecserver:~#
```

To customize their use, we edit the server configuration /etc/ssh/sshd_config [docs.ssh.com, ConfiguringMACalgorithms]. We will test setting the MAC to ‘hmac-sha1’. Then, to apply the changes the service must be restart (`service ssh restart`).

```
# Example of overriding settings on a per-user basis
#Match User anoncvs
#       X11Forwarding no
#       AllowTcpForwarding no
#       PermitTTY no
#       ForceCommand cvs server

MACs            hmac-sha1
"/etc/ssh/sshd_config" 124L, 3254B
```

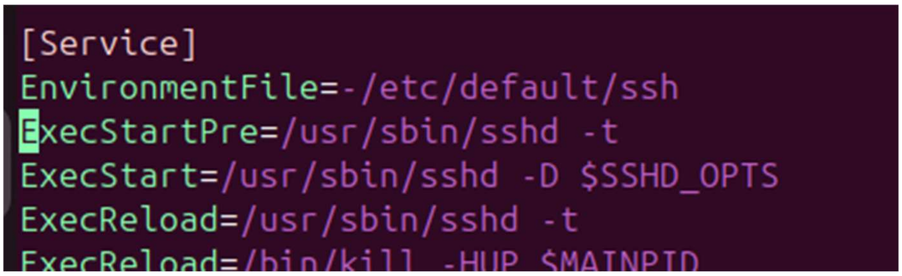


The only listed MAC algorithm is indeed hmac-sha1

```
root@syssecserver:~# vim /etc/ssh/sshd_conf
root@syssecserver:~# sshd -T | grep macs
macs hmac-sha1
root@syssecserver:~#
```

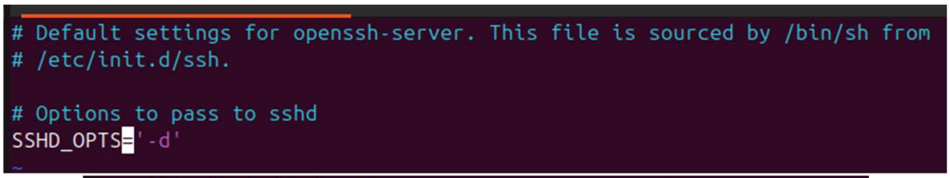
Attempting to attach strace to the ssh daemon normally won’t reveal any sensitive information when a client connects. This is because modern versions of OpenSSH actually handle passwords securely through mechanisms like secure memory buffers, preventing passwords being visible in system calls. The workaround is to launch the daemon in debug mode.

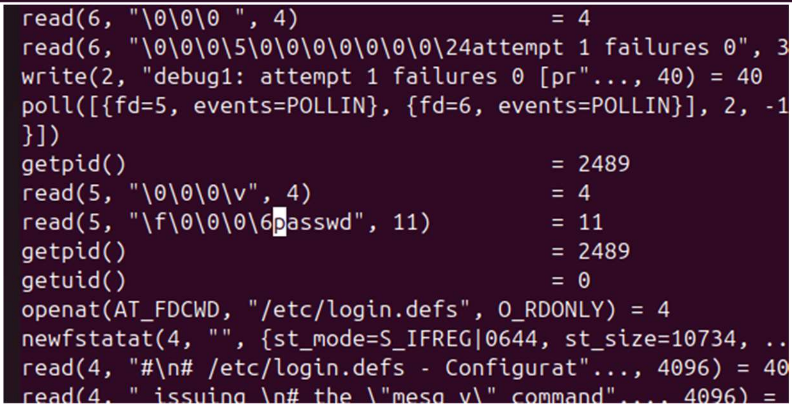
For this purpose we must explore the service config file /etc/systemd/system/sshd.service [askubuntu.com, how-do-i-pass-flags-when-starting-a-service].



On the line "EnvironmentFile" we find the location for the default settings of openssh server where we will add the '-d' debug mode flag as option to pass to sshd startup. Then, we restart the service, find its PID and attach strace to it:

```
$ strace -p $pid -o strace_output.txt
```





strace_output.txt; Here is a read system call with the client's password

5. Using a SSH authority and a server certificate

To generate the CA private-public key pair on the server,acting as it's own CA:

```
$ cd /etc/ssh && ssh-keygen -t rsa -b 4096 -f ca_key
```

Then, to create a certificate [openbsd.org/ssh-keygen]:

```
$ ssh-keygen -s $path_ca_key -n $domain_or_ip -I "My server cert" -V +52w -h /path/to/host_key.pub
$ # -s specifies the private CA key to sign
$ # -h when signing, indicates this is a host certificate and not a user certificate; host certificate authenticates server hosts to users
$ #-n specify one or more principals to be included in the certificate
$ # -I specify identity
$ # -V validity period
```

The resulting certificate will be created at `/path/to/host_key-cert.pub`.

```
root@syssecserver:/etc/ssh# ssh-keygen -s ca_key -h -n 10.0.3.57 -I "My server cert" -V
 +52w -h ssh_host_rsa_key.pub
Signed host key ssh_host_rsa_key-cert.pub: id "My server cert" serial 0 for 10.0.3.57 v
alid from 2024-11-15T15:01:00 to 2025-11-14T15:02:48
root@syssecserver:/etc/ssh# ls -l
total 560
-rw------- 1 root root   3381 Nov 15 14:47 ca_key
-rw-r--r-- 1 root root    743 Nov 15 14:47 ca_key.pub
-rw-r--r-- 1 root root 505426 Jun 26 13:11 moduli
-rw-r--r-- 1 root root   1650 Jun 26 13:11 ssh_config
drwxr-xr-x 2 root root   4096 Jun 26 13:11 ssh_config.d
-rw-r--r-- 1 root root   3254 Nov 15 13:14 sshd_config
drwxr-xr-x 2 root root   4096 Jun 26 13:11 sshd_config.d
-rw------- 1 root root   1381 Nov 14 14:54 ssh_host_dsa_key
-rw-r--r-- 1 root root    607 Nov 14 14:54 ssh_host_dsa_key.pub
-rw------- 1 root root    513 Nov 12 17:42 ssh_host_ecdsa_key
-rw-r--r-- 1 root root    179 Nov 12 17:42 ssh_host_ecdsa_key.pub
-rw------- 1 root root    411 Nov 12 17:42 ssh_host_ed25519_key
-rw-r--r-- 1 root root     99 Nov 12 17:42 ssh_host_ed25519_key.pub
-rw------- 1 root root   2602 Nov 12 17:42 ssh_host_rsa_key
-rw-r--r-- 1 root root   2200 Nov 15 15:02 ssh_host_rsa_key-cert.pub
-rw-r--r-- 1 root root    571 Nov 12 17:42 ssh_host_rsa_key.pub
-rw-r--r-- 1 root root    342 Dec  7  2020 ssh_import_id
```

Next step is to modify sshd_config by adding the following line, then restarting the service:

```
HostCertificate /path/to/host_key-cert.pub
```

```
#HostKey /etc/ssh/ssh_host_rsa_key
#HostKey /etc/ssh/ssh_host_ecdsa_key
#HostKey /etc/ssh/ssh_host_ed25519_key
HostCertificate /etc/ssh/ssh_host_rsa_key-cert.pub
```

On the client, the CA’s public key must be added to a user’s known_hosts with the `@cert-authority` marker. In our environment, we will be copy-pasting the ca_key.pub content from the server container to a ca_key.pub file on the client container. Then, on the client:

```
$ echo "@cert-authority * $(cat ca_key.pub)" >> ~/.ssh/known_hosts
```

In our experience, cancelling by CTRL-C the first login attempt to the sshd service ran in debug mode makes it crash (unhandled event 12). So before moving forward, `/etc/default/ssh` must be modified, removing the flag from `SSHD_OPTS`.

We’ve also tried removing all content before appending the cert authority to `known_hosts`. When connecting via `ssh -v user@domain`, the server will be authenticated. If the CA’s private key (`ca_key`) is leaked, any attacker could generate new host certificates for malicious servers and thus impersonate the legitimate ones (man-in-the-middle attacks). Always have strong file permissions, and encrypt private keys.

```
debug1: SSH2_MSG_KEX_ECDH_REPLY received
debug1: Server host certificate: ssh-rsa-cert-v01@openssh.com SHA256:5tTWXVbnJ5phjqtaXp
C/89RepuHHJOq4NTWXI7saTTE, serial 0 ID "My server cert" CA ssh-rsa SHA256:sVBWT7SA843wa
iDW8ZAoaC9RX5tRS4CMDO4ddPxhM3Y valid from 2024-11-15T15:01:00 to 2025-11-14T15:02:48
debug1: load_hostkeys: fopen /root/.ssh/known_hosts2: No such file or directory
debug1: load_hostkeys: fopen /etc/ssh/ssh_known_hosts: No such file or directory
debug1: load_hostkeys: fopen /etc/ssh/ssh_known_hosts2: No such file or directory
debug1: Host '10.0.3.57' is known and matches the RSA-CERT host certificate.
debug1: Found CA key in /root/.ssh/known_hosts:2
debug1: ssh_packet_send2_wrapped: resetting send seqnr 3
debug1: rekey out after 134217728 blocks
```

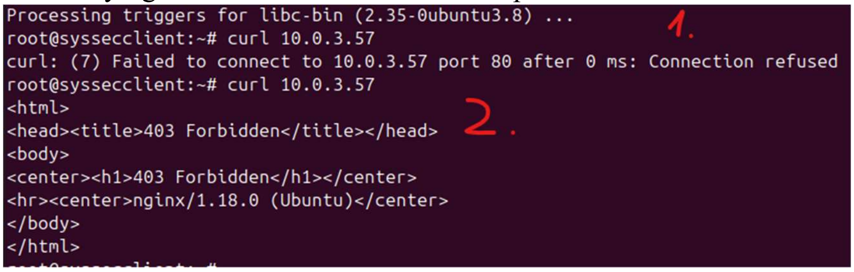
6. Port forwarding

We will be installing a Nginx server with default configuration. For this, on the server container:

```
$ apt install nginx -y
```

To restrict access only to the loopback interface we have 2 possibilities:

- restrict lower on the stack, by using the ‘listen 127.0.0.1 80’ directive in the default server’s config file `/etc/nginx/sites-available/default`; an outside connection is not possible at all
- use ‘`allow 127.0.0.1`’ and ‘`deny all`’ directives, sending HTTP ‘access denied’ error codes to clients trying to access from outside the loopback interface



```
Processing triggers for libc-bin (2.35-0ubuntu3.8) ...          1.
root@syssecclient:~# curl 10.0.3.57
curl: (7) Failed to connect to 10.0.3.57 port 80 after 0 ms: Connection refused
root@syssecclient:~# curl 10.0.3.57
<html>
<head><title>403 Forbidden</title></head>    2.
<body>
<center><h1>403 Forbidden</h1></center>
<hr><center>nginx/1.18.0 (Ubuntu)</center>
</body>
</html>
```

Client attempting to access the nginx web page, blocked through the 2 methods

If we are to use SSH port forwarding, on the client machine we will tunnel the local port 8080 to the nginx server’s loopback port 80, making it persistent, but without a shell:

```
$ ssh -N -L 8080:127.0.0.1:80 user@domain
$ # -N do not execute a remote command; useful for just forwarding ports (ssh v2 only)
$ # -L specifies the given port on the local host to be forwarded to the given host and port on the remote side
```

```
root@syssecclient:~# ssh -N -L 8080:127.0.0.1:80 alex@10.0.3.57
alex@10.0.3.57's password:
^Z
[1]+  Stopped                 ssh -N -L 8080:127.0.0.1:80 alex@10.0.3.57
root@syssecclient:~# bg
[1]+ ssh -N -L 8080:127.0.0.1:80 alex@10.0.3.57 &
root@syssecclient:~# curl 127.0.0.1:8080
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
```

To disable this feature from the ssh server, add or modify the following directive in sshd_config, then restart the service:

```
AllowTcpForwarding no
```

Note that disabling TCP forwarding does not improve security unless users would also be denied shell access, as the they can always install their own forwarders [linux.die.net/man/5/sshd_config]. Another option, though by default is no, is `PermitTunnel`.

For a better security measure, ‘`Match User <username>`‘ blocks in the sshd_config can be used to enforce restrictions for specific users or groups.

```
root@sysecclient:~# ssh -N -L 8080:127.0.0.1:80 alex@10.0.3.57
alex@10.0.3.57's password:
^Z[1]   Killed                  ssh -N -L 8080:127.0.0.1:80 alex@10.0.3.57

[2]+  Stopped                 ssh -N -L 8080:127.0.0.1:80 alex@10.0.3.57
root@sysecclient:~# bg
[2]+ ssh -N -L 8080:127.0.0.1:80 alex@10.0.3.57 &
root@sysecclient:~# ps aux | grep ssh
root        1298  0.0  0.2  16928  8320 pts/5    S    21:56   0:00 ssh -N -L 8080:127.0
.0.1:80 alex@10.0.3.57
root        1300  0.0  0.0   9080  2432 pts/5    S+   21:56   0:00 grep --color=auto ss
h
root@sysecclient:~# curl 127.0.0.1:8080
channel 2: open failed: administratively prohibited: open failed
curl: (56) Recv failure: Connection reset by peer
root@sysecclient:~#
```

Client can launch a ssh connection attempting to forward ports, but access is prohibited

7. Client RSA-key authentication

For the client to generate a key pair, `id_test_ssh` and `id_test_ssh.pub`,

```
$ ssh-keygen -t rsa -b 4096 -f id_test_ssh
```

The private key is stored in plain text, encoded using base64; optionally, it can be symmetrically encrypted (ex. AES) with a passphrase (prompted during generation). Permissions for the private key are set so that only the owner can read it and modify it.

```
root@sysecclient:~# ssh-keygen -t rsa -b 4096 -f id_test_ssh
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in id_test_ssh
Your public key has been saved in id_test_ssh.pub
The key fingerprint is:
SHA256:9OmSC1382KBodgJFyutcbOy29YDgzc+M9NWcoTtUOf4 root@sysecclient
The key's randomart image is:
+---[RSA 4096]----+
|       .         |
|    . o          |
|     o . .    .  |
|      = . o .+   |
|     + = S *o..  |
|    + O + =.O.o  |
|     + & O.= *.  |
|      = % *..  E |
|       o * o.    |
+----[SHA256]-----+
root@sysecclient:~# ls -l
total 24
-rw-r--r-- 1 root    root      743 Nov 15 15:12 ca_key.pub
-rw------- 1 root    root     3381 Nov 15 22:04 id_test_ssh
-rw-r--r-- 1 root    root      743 Nov 15 22:04 id_test_ssh.pub
-rw-r--r-- 1 tcpdump tcpdump 10876 Nov 14 14:06 telnet_cap.pcap
```

Generating a RSA-4096 key pair with no encryption

To transfer the public key to the server, where it will be stored in the user's home directory, specifically `~/.ssh/authorized_keys`.

```
$ ssh-copy-id -I id_test_ssh.pub user@domain
```

```
root@sysecclient:~# ssh-copy-id -i id_test_ssh.pub alex@10.0.3.57
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "id_test_ssh.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
alex@10.0.3.57's password:
Could not chdir to home directory /home/alex: No such file or directory
sh: 1: cd: can't cd to /home/alex
mkdir: cannot create directory '.ssh': Permission denied
```

Copy of the pubic key fails because the user doesn't have a home directory created

```
root@syssecclient:~# ssh-copy-id -i id_test_ssh.pub alex@10.0.3.57
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "id_test_ssh.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
alex@10.0.3.57's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'alex@10.0.3.57'"
and check to make sure that only the key(s) you wanted were added.
```

After creating a home directory for the user on the server, key installation succeeds

```
root@syssecclient:~# ssh -i id_test_ssh alex@10.0.3.57
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 6.8.0-48-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro
Last login: Fri Nov 15 15:30:20 2024 from 10.0.3.198
$ ls .ssh
authorized_keys
$ cat .ssh/authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAACAQCpnzTq5e4tbeZY+f0ubIipf3zpCj0/Jb7IVM/erklz16Uo9W4
/bHK/7fJ8CZgi9q1DXlYHyrvTbknhcokAFt/WJvvsfLNUaPKXHkJSz673X87NTMfv4kxcft2foZDX6nAgF9S7ii
I711Y1cCMS7vJnwkJFC+HfLQ3ZRusVos2Vv5011sR2DTk0g4OOXM+HI+YkmsYz+itnyw3/aieP0V6dWd9UDPS1m
QWPBmkDj2+1zEO7osdld+goPLevLIu0NjUzr+46XP5owRRmbrIJwTybZl4oobFHDGuNBYcDC1HbX5LhrOHPXha/
JHOZHUKmdfbxoZnL+HYKllvmJ0OKMI26AMD2t3Fgs8XVMazjRsw/EkCbjVBS5v9Z9wRB0YwVxA+KoSRU2xoqbHB
lG9KYe218YTJZLaLnEkeEhE55sBKwgwvxTdCuqsjzXCGXJVXgE87k6KxDxTVMtiQ//sR2lJKV8Ib+IMiBgd74/k
xIN907gDlLmAbBtJfJy+6LrqyhDCsfkKU5pmTsYnyGuZB1wc5LPxr0WJzviRW1OTEbBDZpo6dnbcklGdcmA0HYl
t889edIlA2Ui7bpjq2Kok2Ab1GuY8wl3zHEvkDeee1e8w47Fd+f52YhNZpnywu/jiVGCc0e/DMJnkgK9q4wCTn8
z8Lcfa7tzA9lRcnHXpGOP4VMhw== root@syssecclient
```

If RSA authentication would fail, SSH falls back to another enabled authentication method (password authentication, for example), and if no other method succeeds, the connection is denied. RSA authentication provides a better level of security if access to the private key is controlled through proper permissions and if it is encrypted (through a strong passphrase). The public key must be stored only on trusted servers. This method prevents brute-forcing and credential interception as compared to password authentication.

An user on the server with only his access rights cannot disable password authentication, leaving only RSA authentication allowed for his account. Explored options were to configure sshd_config with a `Match User` block, but this requires administrative privileges, and to lock the password of the account – again, administrative privileges required.

`StrictModes` is a setting in `/etc/ssh/sshd_config` that indicates to the SSH server to check the permissions of critical files and directories before allowing access. By default, it is set to `yes`: it verifies that `.ssh` directories and files (for example, `authorized_keys`) are owned by the correct user and that access is not overly permissive. If disabled, SSH will ignore permission issues – it will allow misconfigured files (ex. Others can write to `authorized_keys`).