Using SSH Certificates

Jun 6, 2020 • Roger Ferrer Ibáñez • ssh

Password-based authentication has a number of drawbacks, so many services (such as github) use SSH keys to authenticate. However distributing the keys over several nodes (be virtual machines or single-board computers such as Raspberry Pi) doesn't scale over the number of nodes and users.

Luckily, OpenSSH implementation of SSH supports a certificate-based mechanism. This mechanism may help reducing the complexity of users trusting SSH hosts and hosts trusting SSH users.

Before we continue, a caveat

> Using SSH certificates is not a security panacea. Like every other technological solution it has pros and cons that have to be gauged against the existing requirements.

Concepts

Public key cryptography (I'd dare to say cryptography in general) is notoriously confusing because it uses several terms at the same time that often are at the same "semantic" level and so they are easy to mix up.

In this post we will use the following terminology:

SSH key	A SSH key is a cryptographic widget made up of two keys (each one stored in a different file): the public SSH key and the private SSH key. These two keys are related mathematically but deriving one from the other is not possible.
public SSH key	The public key of a SSH key is the part that can be disclosed and distributed.
private SSH key	The private key of a SSH key is the part that should never be disclosed or distributed.
Certificate Authority	Entity which has its own SSH key (i.e. a public SSH key and a private SSH key) which will be used to emit certificates that we can trust.
certificate	Digital signature issued by a Certificate Authority that asserts the

	authenticity of something, such a SSH key. If the Certificate Authority is trusted we can trust the certificate.
Host	The host is the machine we want to connect to using SSH.
Host Key	Each host has its own SSH key (again, a public and private one) which is used to identify the host. The public key is presented to a user connecting to the host.
User Key	Each user can have one (or more than one) SSH key(s). These keys are used to authenticate the user against each host.

In most scenarios users need to copy (using `ssh-copy-id` or similar) their public SSH key to each host. Auhtentication proceeds by a challenge mechanism. We need to prove that the user has the private SSH key related to the public SSH key found in the host. So the host encrypts a challenge using the public SSH key to be decrypted by the user. The user decrypts the challenge the private SSH key and sends that to the host. If the user had the right private key, the challenge suceeds and no password is required.

However, the first time a user connects to a host, SSH asks if we really want to trust the host. The rationale here is that we might be fooled to login somehwere other than the real host (which could be abused, among others, to disclose our password while trying to login).

We want to make the hosts trusted by the user (imagine a new host is set up) but also we want the hosts trust the user.

When trust is involved a Certificate Authority is required so we will need to create one so the users can trust the hosts and one so the hosts can trust the users.

Host Certificate Authority (Host CA)	A SSH key. Its private SSH key will be used to issue host certificates. Its public SSH key will be used by users to verify that host certificates were issued by the Host Certificate Authority they trust.
User Certificate Authority (User CA)	A SSH key. Its private SSH key will be used to issue user certificates. Its public SSH key will be used by hosts to verify that user certificates were issued by the User Certificate Authority they trust.

Host Certificate Authority

A certificate authority is a big sounding name, but in SSH it is going to be just another SSH key (again with public SSH key and its private SSH key).

We will be using `ssh-keygen` all the time so be careful with the flags. First we need to create the SSH key that will act as the Host Certificate Authority.

```
$ ssh-keygen -t ed25519 -f host_ca
```

This will generate two files: `host_ca` is the private SSH key and `host_ca.pub` is the public SSH key. The private key is trusted and so it is essential to keep it super safe. If you use another name in the `-f` option, say `-f my_host_ca`, the created files will be `my_host_ca` and `my_host_ca.pub`, respectively.

If you look at the contents of the public key

```
$ cat host_ca.pub
ssh-ed25519 AAAA... comment
```

The first part `ssh-ed25519` is the kind of SSH key. A key of kind `ed25591` seems to be recommended over the default `rsa`, hence the flag `-t ed25519` earlier. The `AAAA...` part is specific to your SSH key and `comment` is usually `user@host` but can be changed to be more informative with an option `-C comment` in the `ssh-keygen` call above.

Set a user to trust the Host CA

In order to make a user trust our new Host CA, we need to distribute the **public** SSH key of the Host CA and tell `ssh` to trust it.

This can be used system-wide (i.e. the computer of the user) or per user in that system.

Let's assume we want the node `UserMachine1` trust the Host CA.

- If we want the trust relationship be system-wide, the file to edit is `/etc/ssh/ssh_known_hosts` of `UserMachine1`. In general only administrators can edit this file.
- If a user in `UserMachine1` wants to individually trust that `Host CA` the file to edit is `$HOME/.ssh/ssh_known_hosts`.

In either case we have to add the following line.

```
@cert-authority *.example.com ssh-ed25519 AAAA...
```

Here `*.example.com` will be your domain name (it is very handy to have your own domain name even in your LAN at home either using ISC BIND and ISC DHCP or using dnsmasq) for which the certificate will be trusted. The `ssh-ed25519 AAAA...` part is the contents of the public SSH key of the Host CA.

And that's it. From now on when trying to connect to a host that matches `*.example.com`, if that host presents a certificate that has been signed by the Host CA we trust, we will also trust the host.

So the next logical step is how to create a host certificate.

Issue a Host Certificate

To issue a host certificate we need the host public key of the host. This key is found in `/etc/ssh/ssh_host_ed25519_key.pub`.

Imagine we have a host `ServerMachine1.example.com`. What we have to do is to obtain its host public SSH key. We have to copy it (over the network or via a USB if we've got physical access) to the place where we have our `host_ca` (the private SSH key of the Host CA).

Now we need to emit a host certificate. We do that again using `ssh-keygen`.

```
$ ssh-keygen -s host_ca \
    -I "ServerMachine1" \
    -h \
    -n ServerMachine1.example.com \
    -V +52w \
    ssh_host_ed25519_key.pub
```

There are many flags here so let's see each one:

- `-s host_ca` means issuing a certificate using the `host_ca` private SSH key.
- `-I "ServerMachine1"` is an arbitrary identifier we can use to help us identify the key in the server logs.
- `-h` means create a host certificate
- `-n ServerMachine1.example.com` is the set of host names (comma-separated) for which this certificate is issued.
- `-V +52w` means that this certificate will be valid for 52 week, after that time the certificate won't be trusted. If this option is not specified, the issued certificate will be valid forever.
- `ssh_host_ed25519_key.pub` this is the host public SSH key for which we are issuing a certificate (copied from `ServerMachine1.example.com`).

This will generate a file `ssh_host_ed25519_key-cert.pub` (note the `-cert` before `.pub`). This file now has to be copied back to `ServerMachine1.example.com`. For instance we can copy it into `/etc/ssh/ssh_host_ed25519_key-cert.pub`.

Now we need to tell the SSH server in `ServerMachine1.example.com` to offer that certificate to all incoming SSH connections. We can do that editing the file `/etc/ssh/sshd_config` and adding the following line (in general only administrators can change this file).

```
HostCertificate /etc/ssh/ssh_host_ed25519_key-cert.pub
```

Now we have to restart the SSH server. In mosts systems with `systemd` an administrator can do that by running the following comment.

```
$ sudo systemctl restart ssh
```

Testing the Host Certificate

Now a user that trusts the Host Certificate Authority should be able to trust without having to do anything. We already established that trust relationship in `UserMachine1` so let's use that machine to test it.

First make sure no trust relationship is remembered in `UserMachine1`.

```
$ ssh-keygen -R ServerMachine1.example.com
```

Still from `UserMachine1` try to to login to `ServerMachine1.example.com`

```
$ ssh ServerMachine1.example.com
```

If you see something like this

```
The authenticity of host 'ServerMachine.example.com (1.2.3.4)' can't be establis
RSA key fingerprint is SHA256:aeghei4Cao0quuteij3aechu.
Are you sure you want to continue connecting (yes/no)?
```

something is wrong and you need to recheck the steps above.

If you're directly requested the password (or some previous SSH key allows you to login) then `UserMachine1` is trusting `ServerMachine1`.

Note that some organizations add domain suffixes when solving names. So `ServerMachine1` can be used to access `ServerMachine1.example.com`. However SSH is strict about names and `ssh ServerMachine1` will not trust the machine.

If you want to use the short name (or any other name) my recommendation is to teach ssh in `UserMachine1` that `ServerMachine1` is actually `ServerMachine1.example.com`. You can do that adding the following lines in `$HOME/.ssh/config`

```
Host ServerName1
  HostName ServerName1.example.com
```

User Certificate Authority

Now that we trust the servers in our organization, the next step is that the servers trust our users.

To do that we need to create a user certificate authority.

```
$ ssh-keygen -t ed25519 -f user_ca
```

Again this will generate a `user_ca` file with the private SSH key that must be kept safe and secret, and a `user_ca.pub` that contains the public SSH key.

Set a server to trust the User CA

A server like `ServerMachine1.example.com` needs to trust the User CA before it can trust any certificate issued by that User CA.

We do this copying the file `user_ca.pub` to `ServerMachine1`. For instance in `/etc/ssh/user_ca.pub`. Now we edit the file `/etc/ssh/sshd_config` and we add the following lines.

```
TrustedUserCAKeys /etc/ssh/user_ca.pub
```

Now we need to restart the SSH server.

```
$ sudo systemctl restart ssh
```

Now the SSH server in `ServerMachine1.example.com` trusts the user certificates issued by the User CA we created in the earlier section.

Issue a User Certificate

To issue a user certificate, we need the public SSH key of a user. I'm assuming that the user already has one in `$HOME/.ssh/id_ed25519.pub` in `UserMachine1`.

Copy the public SSH key of the user where the `user_ca` private SSH of the User CA is found.

Now use the following command

```
$ ssh-keygen -s user_ca \
      -I "user_name" \
      -n userid1 \
      -V +52w \
      id_ed25519.pub
```

Let's review each flag

- `-s user_ca` means issuing a certificate using the `user_ca` private SSH key.
- `-I "user_name"` is an arbitrary identifier we can use to help us identify the key in the server logs.
- `-n userid1` is the set of user identifiers (comma-separated) for which this certificate is issued. This is the userid used to login.
- `-V +52w` means that this certificate will be valid for 52 week, after that time the certificate won't be trusted. If this option is not specified, the issued certificate will be valid forever.
- `id_ed25519_key.pub` this is the user public SSH key for which we are issuing a certificate (the user must provide it).

This will create a file `id_ed25519_key-cert.pub` (mind the `-cert` before `.pub`) that we have to copy back to `UserMachine1`. It has to be copied in `$HOME/.ssh/id_ed25519-cert.pub` (be careful not to overwrite the public SSH key).

Testing the User Certificate

Note the user should be able to login to `ServerMachine1`.

```
$ ssh ServerMachine1.example.com
```

It is necessary that no password is requested otherwise something is wrong.

It is not sufficient however. A key in `ServerMachine1`, found in `$HOME/.ssh/authorized_keys` could still be being used to allow us to login.

Make sure `$HOME/.ssh/authorized_keys` in `ServerMachine1` does not contain any public key of the user of `UserMachine1`. If there is one it might be allowing password-less login instead of our certificate. So, make sure no key from `UserMachine1` is listed in `$HOME/.ssh/authorized_keys` of `ServerMachine` (a more radical alternative is to remove `$HOME/.ssh/authorized_keys` of `ServerMachine1`). After that, you should be able to login without being requested a password, otherwise recheck the steps above.

« Fortran and modules   Create your own local domain and DHCP with dnsmasq »