sshca—tool to manage SSH CA certificates

Lars Wirzenius

2022-06-18 09:23

Contents

Chapter 1

Introduction

The `sshca` tool helps manage an SSH Certificate Authority (CA) and create host and user certificates. Such certificates make using and administering SSH less tedious and more secure, by removing the need for users to check host keys, or maintain `authorized_keys` files.

An SSH CA is an SSH key dedicated to signing, or certifying, other SSH keys. Such a signed key is called a certificate and is used together with the private part of the certified key. The certificate is used instead of the public key.

SSH clients and servers can be configured to trust certificates made by one or more CA keys. This makes it possible for a client to trust a server without asking the user to accept the host key for each new server. A server can trust a client without having the client's public key configured for that user in the `authorized_key` file. This simplifies overall key management significantly, but requires creating and managing CA keys and certificates.

1.1 Host certificates

Traditionally, in the world of SSH, servers have host keys that rarely change, but are generated separately for each host. When a user accesses a host for the first time, at a given address, they are presented with the host's public key, and need to manually, laboriously, and usually insecurely, check that it's the right key for that host.

This can be a risky situation: if an attacker manages to trick the user's SSH client to show a key the attacker has generated, and the user accepts it as the real one, the attacker can see – and change – all the traffic going over the SSH connection. This mostly nullifies the security benefit SSH is meant to provide. (Not entirely: the connection is still protected against other attackers.)

In a situation where there are many hosts, or hosts gets recreated often, or change address a lot, all of which happen when using cloud technologies, the risky situation keeps happening frequently. Not only is it risky, it is also tedious and cumbersome to the user. If this keeps happening a lot, users are in effect trained to automatically accept all host keys. This is an example of bad usability being bad security.

The risky situation can be avoided by having the host keys be communicated to all users ahead of time, but doing this in a secure and convenient way is difficult. It is also unnecessary.

Using an SSH CA to certify SSH host keys means the user's SSH client can trust it without asking the user to verify it. The client is configured to trust any host certificate that can be verified using the SSH CA public key. The CA public key still needs to be communicated to the user in a secure way, but the CA key is only one key and rarely changes, so the tiresome risky situation happens very rarely. After the user has the CA key, an attacker can't trick the user into accepting a false host key.

With host certificates, the SSH client never needs to ask its user if the host key of a new host is valid, and the user never needs to try to verify it. If the host's identity (host key or address) changes, such as when a virtual machine is re-created, the client doesn't need to bother the user about it, as long as the new identity gets a new certificate.

Overall, this leads to a much smoother and more secure experience for people using SSH.

1.2 User certificates

Traditionally, a user authenticates themselves to an SSH server using a password, or a user key. We will not discuss passwords here. You should not use passwords. Your SSH server should not accept passwords.

A user SSH key is an SSH key pair of which the user has the private part. The public part is added to the `~/.ssh/authorized_keys` file on the server for the user's account. At login time, the client proves to the server that it has the private key that corresponds to one of the public keys in that file, and this proves to the server that the user is authorized to log in. This is great, because the user does not need to remember a strong password, nor type it in every time they log in, and the server does not need to store the user's password at all, even in an encrypted way.

The result is an easy, secure way for the user to log into the server. However, this only works if the list of authorized keys is kept up to date.

If the user needs to, for any reason, change to a new key, perhaps as part of a regular key rotation strategy, the list of authorized keys needs to be updated to add the new key, and remove the old key. This needs to be done on each server the user uses. The update procedure is often a risky, tedious step. If an attacker manages to get the attacker's key into the list, they can log into the server as the user. Given that the `authorized_keys` file is usually user-editable, the user may add any SSH public keys to that file, including keys for other people, or keys stored on machines that are insecure. The user may do this intentionally, or because they've been tricked or coerced into doing it.

An SSH CA can create a user certificate, which ties a user's SSH public key to a username. An SSH server can be configured to trust such certificates, made with specific CA keys, and to act as if that user's public key is in their `authorized_keys` file, even if that file doesn't exist. The result is that there is no need to maintain that file. This also means it's feasible to revoke access with specific certificates.

The user certificate replaces the public key in the SSH authentication process. The user still needs the corresponding private key to authenticate: the certificate itself is not enough.

Overall, this leads to system administrators having an easier way to control who has access their servers over SSH.

1.3 Certificate automation

Generating all these certificates can be done using the `ssh-keygen` command line tool. However, it's just intricate enough that it becomes tedious and cumbersome and thus error prone. The `sshca` tool makes it easier.

1.4 Configuring servers and user accounts

The `sshca` tool does not install host certificates on servers, nor configure servers or user accounts to trust certificates made using specific CA identities. The server system administrators and users need to do that themselves.

1.5 SSH CA vs SSHFP

Another approach is to distribute host keys via DNSSEC using SSHFP DNS records. This requires DNSSEC to work for all clients, and only works for verifying host identities. However, they may be easier to adopt for some organizations.

Chapter 2

Requirements for SSH CA automation

Automation for SSH CA management needs to satisfy all the following high-level requirements to be acceptable.

- **Secure.** SSH is meant to enable use of remote systems and file transfers in a secure manner. SSH CA is meant to improve security further. Any automation of SSH CA must not compromise on security.
- **Convenient to use** for both system administrators and end users. It is a fundamental realization that people will do things in the convenient manner, and security needs to enable that. SSH CA makes use of SSH more convenient, and automation for it must not squander that.
- **Convenient to integrate** with existing SSH management infrastructure.

The following sections document more detailed acceptance criteria and how they are verified in an automated manner.

2.1 Smoke test

This scenario verifies that the `sshca` command line tool can be invoked at all, in the simplest possible ways.

given an installed sshca
when I run **sshca --help**
then stdout contains "**--help**"

2.2 CA key management

It must be possible to manage multiple SSH CA keys. This scenario verifies that `sshca` can create and delete CA keys.

Initially the store must be empty and have no CA keys.

given an installed sshca
and file **.config/sshca/config.yaml** from **config.yaml**
when I run **sshca ca list**
then stdout is exactly ""

File: **config.yaml**

```
store: store.yaml
```

When we create a new CA key, it shows up in the list.

when I run **sshca ca new host hostCAv1**
and I run **sshca ca list**
then stdout contains "**hostCAv1**"

We can see the CA public key.

when I run **sshca ca public-key hostCAv1**
then stdout matches regex **^ssh-ed25519\s\S+\s$**

When we remove a CA key, it's no longer in the store.

when I run **sshca ca delete hostCAv1**
and I run **sshca ca list**
then stdout is exactly ""

When we create two CA keys, they can be individually removed.

when I run **sshca ca new host CAv1**
and I run **sshca ca new host CAv2**
and I run **sshca ca list**
then stdout contains "**CAv1**"
and stdout contains "**CAv2**"
when I run **sshca ca delete CAv1**
and I run **sshca ca list**
then stdout doesn't contain "**CAv1**"
and stdout contains "**CAv2**"
when I run **sshca ca delete CAv2**
and I run **sshca ca list**
then stdout is exactly ""

2.3 Host certificate management

It must be possible to generate host certificates. Once a host key is imported, it must be possible to generate new host certificates for it.

Initially, there must be no host keys imported.

given an installed sshca
and file **.config/sshca/config.yaml** from **config.yaml**
when I run **sshca host list**
then stdout is exactly ""

We must be able to import a host key.

when I run **ssh-keygen -t ed25519 -N ” -f myhost**
and I run **sshca host new myhost.example.com myhost.pub**
and I run **sshca host list**
then stdout contains "**myhost.example.com**"

Importing a key with a name that is already in use must fail.

when I run **ssh-keygen -f myhost2 -N ""**
and I try to run **sshca host new myhost.example.com myhost2.pub**
then command fails
and stderr contains "**myhost.example.com**"
when I run **sshca host list**
then stdout contains "**myhost.example.com**"

We must be able to certify a host.

when I run **sshca ca new host CAv1**
and I run **sshca host certify CAv1 myhost.example.com**
then stdout matches regex **ˆssh-ed25519-cert-v01@openssh.com**

We must be able to remove an imported host key.

when I run **sshca host remove myhost.example.com**
and I run **sshca host list**
then stdout is exactly ""

2.4 User certificate management

It must be possible to generate user certificates. Once a user key is imported, it must be possible to generate new user certificates for it.

Initially, there must be no user keys imported.

given an installed sshca
and file **.config/sshca/config.yaml** from **config.yaml**
when I run **sshca user list**
then stdout is exactly ""

We must be able to import a user key.

when I run **ssh-keygen -t ed25519 -f myself -N ""**
and I run **sshca user new myself myself.pub**
and I run **sshca user list**
then stdout contains "**myself**"

Importing a key with a name that is already in use must fail.

when I run **ssh-keygen -f myself2 -N ""**
and I try to run **sshca user new myself myself2.pub**
then command fails
and stderr contains "**myself**"
when I run **sshca user list**
then stdout contains "**myself**"

We must be able to certify a user.

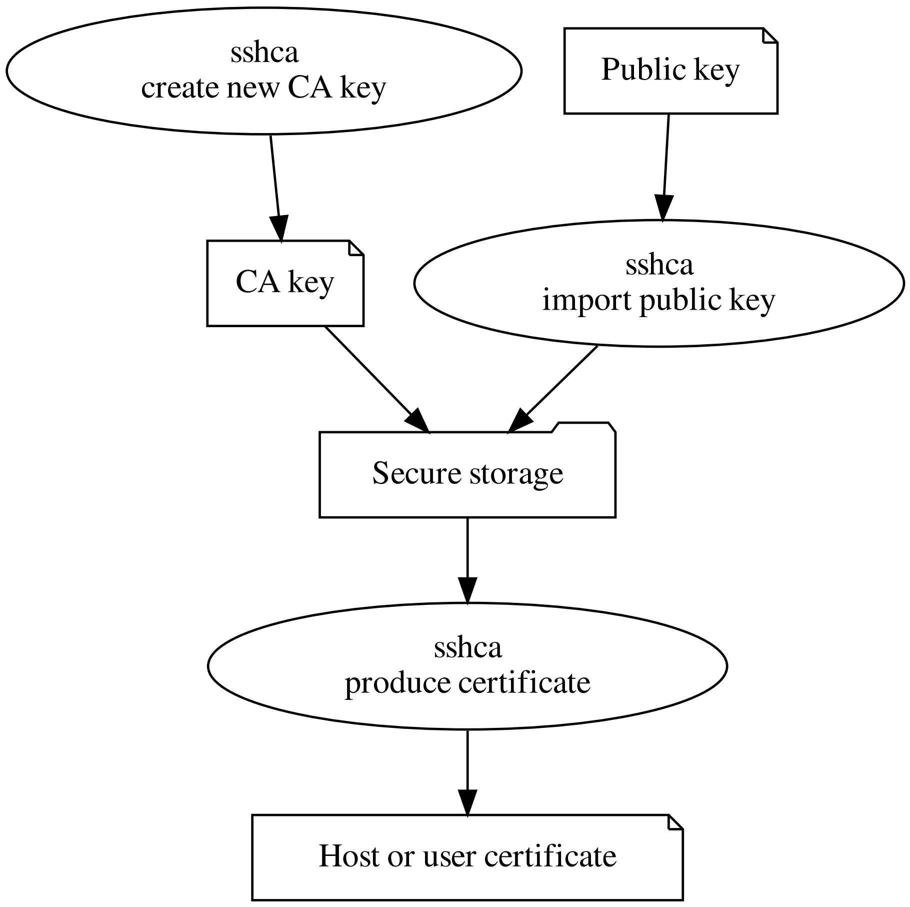
when I run **sshca ca new user CAv1**
and I run **sshca user certify CAv1 myself**
then stdout matches regex **^ssh-ed25519-cert-v01@openssh.com**

We must be able to remove an imported host key.

when I run **sshca user remove myself**
and I run **sshca user list**
then stdout is exactly ""

Chapter 3

The `sshca` command line tool



The `sshca` tool maintains a secure storage of CA key pairs, and host and user public keys, and can use the CA keys and the stored public keys to generate host and user certificates.

3.1 The store

Security note: The `sshca` tool maintains a *store* of SSH public and private keys, as a directory on the local file system. This store is assumed to be trusted: any key there is assumed to have been vetted before being added. The user of the tool should ensure the store can only be accessed by them and not by other parties. The security and integrity of the SSH CA system maintained by `sshca` depends on that.

The store is kept in `~/.local/state/sshca` by default, but the location can be configured via the tool configuration file.

3.2 CA identity management

- `sshca new-ca KIND NAME`

  Create a new CA identity with a given name, and store in the `sshca` store. The name must not be in use yet. `KIND` must be "`user`" or "`host`" for user or host certificates, respectively. The new CA can only be used for indicated kind of certificate.

  CA keys are always of `ed25519` type. This type of key is short, and can certify any type of key.

- `sshca list-ca`

  List known CA identities in the store and their names and whether they're for host or user certificates.

- `sshca delete-ca NAME`

  Remove an existing CA identity from the store.

- `sshca audit-log` — **NOT IMPLEMENTED YET**

  List all the actions the tool has done: every CA creation and deletion, as well as imports, deletions, and renames of host and user public keys, and also any certificates that have been created.

3.3 Host certificate management

- `sshca import-host-key FILE HOSTNAME...`

Import a host public key into the store, and assign it the list of host names given. At least one host name must be given. The host names must be new and not yet used by any host.

- `sshca list-host-keys`

  List all host keys in the store. For each key, show all host names its been given, and also a unique identifier to separate different keys for the same host. This is `KEYID` below. They key identifier is permanent and won't change as long as the key is in the store.

- `sshca certify-host [OPTIONS] CANAME HOSTNAME`

  Create a host certificate for given host public key, using a given CA identity (`CANAME`) for hosts. The certificate will be valid for all the host names associated with the host key, as set when imported or changed with `rename-host-key`, not just for the names given to `certify-host`. The certificate is written to stdout.

  Options can be used to specify the validity time of the certificate: `--valid-from=T` and `--valid-until=T`, where `T` is a timestamp of ISO 8601 form. The `--valid-for=PERIOD` option can be used to specify how long the validity period is: the end time will then be computed by the tool. If neither `--valid-until` nor `--valid-for` is used, the default validity period from the configuration file is used.

  This can be run any number of times. It creates a new certificate every time. The `sshca` tool automatically increments the serial number for each certificate so that they are unique and can be individually revoked.

3.4 User certificate management

- `sshca import-user-key FILE USERNAME COMMENT` — **NOT IMPLEMENTED YET**

  Import a user's public key into the `sshca` store. The username will be used by default when creating a user certificate, and must not be used for another user. The comment is free-form text that describes who the user is. The tool does not interpret it in any way.

- `sshca list-user-keys` — **NOT IMPLEMENTED YET**

  List all users with keys in the store.

- `sshca remove-user-key USERNAME...` — **NOT IMPLEMENTED YET**

  Remove all specified user keys.

- `sshca certify-user [OPTIONS] CANAME USERNAME` — **NOT IMPLEMENTED YET**

Create a user certificate for given user public key, using a given CA identity (`CANAME`). The certificate will be valid for the given user. The certificate is written to stdout.

The same options for controlling the validity period for host certificates can be used here. See `certify-host` above.

This can be run any number of times. It creates a new certificate every time. The `sshca` tool automatically increments the serial number for each certificate so that they are unique and can be individually revoked.

3.5 Revocations

The `sshca` tool does not support revoking certificates. Revocations can be done manually using `ssh-keygen`, but it may be easier to use certificates with short validity periods and creating and distributing new certificates host and users when needed. This is easier for host certificates, which are under direct system administrator control.

For user certificates, a self-serve system would be good, but not currently available. However, the system administrator can generate and publish new user certificates frequently. User certificates are not secret, and they're tightly tied to the user's private key. User certificates are useless without the private key.

3.6 Tool configuration

In `~/.config/sshca/config.yaml` (or other location as specified according to the XDG directory standard), a configuration file can specify:

- `store`

  Fully qualified, tilde-expanded path of the store location.

- `default-validity-duration` — **NOT IMPLEMENTED YET**

  How long a certificate should be valid for by default. An integer that defaults to seconds, but can be suffixed with `h` or `d` for hours or days, respectively. Unless overridden by options, new certificates are valid from the time of creation for the default duration.

Chapter 4

SEE ALSO

- ssh-keygen manual page[1]

[1]https://www.man7.org/linux/man-pages/man1/ssh-keygen.1.html#CERTIFICATES

Chapter 5

Thanks

While writing this, the author got feedback and reviews of drafts from David Leggett.