sigstore Getting Started Guide v1.2

A new standard for signing, verifying and protecting software.

First of all, we recommend you read the "How it Works" page on the sigstore website, as it will help you get familiar with the sigstore projects.

Using Cosign to sign and then verify a container image, storing it in a container registry

1. Install cosign:
   a. If you have Go 1.16+, you can directly install by running:
      ```
      $ go install github.com/sigstore/cosign/cmd/cosign@latest
      ```
   b. Otherwise, you will need to install Go, which is out of scope for this guide.

2. Generate a keypair:

   ```
   $ cosign generate-key-pair
   ```
   You will get a public and a private key (`cosign.pub` and `cosign.key`)

3. Start signing a container image and storing the signature in a container registry:

   ```
   $ cosign sign --key cosign.key $CONTAINER_REGISTRY/$USERNAME/$CONTAINER_IMAGE
   ```

   Hint: if you don't have a registry set up, you can use ttl.sh or, if you have a Github account, you can use Github's container registry service.
   This would result in a command such as:
   `$ cosign sign --key cosign.key https://ttl.sh/my-container-image:1h` or
   `$ cosign sign --key cosign.key https://ghcr.io/axelsimon/my-container-image`

4. You can now verify the image from the container registry:

   ```
   $ cosign verify --key cosign.pub $CONTAINER_REGISTRY/$USERNAME/$CONTAINER_IMAGE
   ```

Hint: to verify an image, Cosign will need to download it first. This will fail if you haven't pushed the image to the registry first. I.e: you may need to run:

`$ podman push ttl.sh/my-container-image:1h` or
`$ podman push ghcr.io/axelsimon/my-container-image`

Note about container registries and podman: Cosign currently looks for your authentication credentials in Docker-specific paths. If you are using podman, you will need to copy the credentials to one of these paths, ie:

```
$ cp $XDG_RUNTIME_DIR/containers/auth.json ~/.docker/config.json
```

Going further: storing other software artefacts, signing and verifying them

Let's create a simple text file describing our container image and then upload it to the container registry. We will then sign it and verify it.

1. Create your text file:

   ```
   $ nano $CONTAINER_IMAGE.txt
   ```

2. Push your text file to the container registry with cosign:

   ```
   $ cosign upload blob -f $CONTAINER_IMAGE.txt
   $CONTAINER_REGISTRY/$USERNAME/$CONTAINER_IMAGE.txt
   ```
   ie: `$ cosign upload blob -f my-container-image.txt ttl.sh/my-container-image.txt:1h`

3. You can also sign the file and upload the signature, as we did for the container image:

   ```
   $ cosign sign --key cosign.key ttl.sh/my-container-image.txt:1h
   ```

   You will get a URL in return, that contains the SHA-256 digest of the artefact.

4. To verify this artifact, you could simply download the file from the container registry (using `curl` or `wget`) and then check its hash, using the SHA-256 digest built into its name to verify it. You would still need to verify the signature manually, however. Thankfully, there is a simpler solution: the `sget` tool. To install it:
   a. `$ go install github.com/sigstore/cosign/cmd/sget@latest`
   b. Then, to verify:

```
      $ sget
ttl.sh/my-container-image.txt@sha256:32983ddbe25c9df2ff5ed08551daa62ceddf527ae441342c
1154252aea4eaf73
```

Note: your container image name and SHA-256 digest will be different, of course.

`sget` automatically verifies an associated signature and integrates with the sigstore binary transparency log, rekor. For more information, refer to the cosign README.

Going further: using sigstore “keyless” signing

In this third step, we will now start making use of sigstore’s main security proposition:”‘keyless” signing. By keyless, we mean that there are no keys to manage, not that there are no keys ever. The signing keys are created and stored in temporary memory, verified and associated with your identity using sigstor’s Fulcio certificate authority, and the results of the checks are stored in Fulcio and in Rekor. To learn more about “keyless” signing, refer to the documentation.

This part of cosign is still experimental, so we will have to set an environment variable to enable it.

1. First, we will sign our image:

   ```
   COSIGN_EXPERIMENTAL=1 cosign sign $CONTAINER_REGISTRY/$USERNAME/$CONTAINER_IMAGE
   ```
   ie: `COSIGN_EXPERIMENTAL=1 cosign sign ttl.sh/my-container-image:1h`

   Cosign will defer to the sigstore service to authenticate you, offering a few identity providers to choose from (more will be available in the future).
   Once you have authenticated with your identity provider, cosign will transparently go through the sigstore signing process, obtaining a certificate from Fulcio, deleting the initial keys, storing the signature in Rekor.

2. Then to verify, we can simply do:

   ```
   COSIGN_EXPERIMENTAL=1 cosign verify $CONTAINER_REGISTRY/$USERNAME/$CONTAINER_IMAGE
   ```
   ie: `COSIGN_EXPERIMENTAL=1 cosign verify ttl.sh/my-container-image:1h`

   And, if the signature checks out, cosign will return the certificate used to sign the image, showing you the email of the original signer.