Cosign Sign and verify artifacts

START HERE

`cosign generate-key-pair`
Make cosign.key and cosign.pub

`cosign sign --key cosign.key <image>@<digest>`
Sign an image

`cosign verify --key cosign.pub <image>`
Check a signature

`cosign sign <image>@<digest>`
Keyless sign via OIDC login

`cosign sign-blob --bundle <b> <file>`
Sign any file

`cosign verify-blob --bundle <b> <file>`
Check a file

`cosign attest --predicate <f> <image>`
Attach signed metadata

`cosign tree <image>`
List sigs, SBOMs, attestations

OIDC login Google, GitHub, ... → `identity token` → **Fulcio** short-lived cert → `sign with ephemeral key` → **Signature** stored in registry → `log entry` → **Rekor** transparency log

Keyless signing

CONCEPTS

Sign by digest (image@sha256:...), not by tag: a tag can move to a different image after you sign.

Keyless verify needs both an expected identity and an expected OIDC issuer; without them anyone with any Sigstore identity would pass.

Signatures and attestations are stored as extra artifacts next to the image in the same registry, so signing needs push access.

KEYS

`cosign generate-key-pair`
Create cosign.key and cosign.pub

`cosign generate-key-pair --output-key-prefix <name>`
Choose the key file names

`cosign generate-key-pair --kms <kms-uri>`
Create the key in a cloud KMS

`cosign public-key --key <kms-uri>`
Export a KMS public key

`--key awskms:// / gcpkms:// / azurekms:// / hashivault://`
Key in a KMS (URI schemes)

SIGN

`cosign sign --key cosign.key <image>@<digest>`
Sign with a local key

`cosign sign <image>@<digest>`
Keyless sign (OIDC login)

`cosign sign --yes <image>@<digest>`
Skip confirmation prompts (CI)

`cosign sign -a <key>=<value> <image>@<digest>`
Add signed annotations

`cosign sign --recursive <image>@<digest>`
Sign each image of a multi-arch index

`cosign sign --identity-token <token> <image>@<digest>`
Use a supplied OIDC token

SIGN (cont.)

`cosign sign --key <kms-uri> <image>@<digest>`
Sign with a KMS key

`cosign sign --signing-config <file> <image>@<digest>`
Use your own Fulcio/Rekor setup

VERIFY

`cosign verify --key cosign.pub <image>`
Verify with a public key

`cosign verify --certificate-identity <id> --certificate-oidc-issuer <url> <image>`
Verify a keyless signature

Use the -regexp variants of both flags to match patterns.

`cosign verify --certificate-identity-regexp <re> --certificate-oidc-issuer-regexp <re> <image>`
Match identity by regexp

`cosign verify -a <key>=<value> --key cosign.pub <image>`
Require matching annotations

`cosign verify --certificate-github-workflow-repository <owner/repo> ... <image>`
Pin the GitHub repo that signed

`cosign verify --key cosign.pub <image> | jq`
Read the verified payload

`cosign verify --key cosign.pub --local-image <dir>`
Verify an image saved to disk

`cosign verify --insecure-ignore-tlog --key cosign.pub <image>`
Skip the transparency log check

Only for signatures never uploaded to the log.

ATTESTATIONS

`cosign attest --key cosign.key --predicate <file> <image>@<digest>`
Attach a signed predicate

`cosign attest --type <type> --predicate <file> <image>@<digest>`
Set the predicate type

Types: slsaprovenance, spdx, spdxjson, cyclonedx, vuln, openvex, custom or a URI.

ATTESTATIONS (cont.)

`cosign verify-attestation --key cosign.pub <image>`
Verify attestations

`cosign verify-attestation --type <type> --policy <file> --key cosign.pub <image>`
Check predicate against a policy

Policy files are CUE or Rego.

`cosign attest --predicate - <image>@<digest>`
Read the predicate from stdin

`cosign verify-attestation ... <image> | jq -r .payload | base64 -d`
Decode the statement

BLOBS

`cosign sign-blob --bundle <bundle> <file>`
Keyless sign a file into a bundle

`cosign verify-blob --bundle <bundle> --certificate-identity <id> --certificate-oidc-issuer <url> <file>`
Verify a keyless file signature

`cosign sign-blob --key cosign.key <file>`
Sign a file with a key

Prints the base64 signature to stdout.

`cosign verify-blob --key cosign.pub --bundle <bundle> <file>`
Verify a file with a public key

`cosign sign-blob --yes --bundle <bundle> <file>`
Non-interactive file signing

`cosign attest-blob --predicate <file> --bundle <bundle> <file>`
Attest a file

INSPECT AND MANAGE

`cosign tree <image>`
Show signatures, SBOMs, attestations

`cosign save --dir <dir> <image>`
Save image and signatures to disk

`cosign load --dir <dir> <image>`
Upload a saved image to a registry

`cosign clean <image>`
Remove signatures from an image

INSPECT AND MANAGE (cont.)

`cosign initialize`
Fetch the Sigstore trust root

`cosign login <registry>`
Log in to a registry

`cosign version`
Print the cosign version

ENVIRONMENT

`COSIGN_PASSWORD=<pw>`
Key password without a prompt

`SIGSTORE_ID_TOKEN=<token>`
Supply an OIDC token, skip login

`SIGSTORE_ROOT_FILE=<pem>`
Use a custom root certificate

`cosign env --show-descriptions`
List all variables cosign reads

In CI (GitHub Actions) keyless signing picks up the workflow's OIDC token automatically; the job needs id-token: write.

RECIPES

Key-based sign and verify

```
cosign generate-key-pair
cosign sign --key cosign.key
ghcr.io/me/app@sha256:<digest>
cosign verify --key cosign.pub
ghcr.io/me/app@sha256:<digest>
```

Keyless in CI, verify by identity

```
cosign sign --yes ghcr.io/me/
app@sha256:<digest>
cosign verify ghcr.io/me/
app@sha256:<digest> \
  --certificate-identity-regexp
'https://github.com/me/app/.*'
\
  --certificate-oidc-issuer
https://token.actions.
githubusercontent.com
```

Sign a release file

```
cosign sign-blob --yes --bundle
app.tgz.sigstore.json app.tgz
cosign verify-blob --bundle
app.tgz.sigstore.json \
  --certificate-identity <id>
--certificate-oidc-issuer <url>
app.tgz
```

Attach and check an SBOM attestation

```
cosign attest --key cosign.key
--type spdxjson --predicate
sbom.spdx.json <image>@<digest>
cosign verify-attestation --key
cosign.pub --type spdxjson
<image>@<digest>
```