Safe Script Retrieval (sget)

Author: Luke Hinds, Jyotsna Penumaka

Goal: There are many instances of curl/wget shell pipe style script executions out there. For all the ease of use they bring, they also expose users to an unacceptable high level of risk (we have had one high profile attack this year already). This proposal extends the existing prototype work carried out within cosign/sget by Dan Lorenc. We introduce keyless style OIDC based policy files and other extra features (such as os/exec operation on validation)

root policy manifest

A root project policy will be used that loosely follows a TUF style `root` structure. This will be mapped to a container registry namespace.

Root file will contain policy elements such as the project's maintainers (those with authority to sign a release), the signing threshold and the project namespace. The `body` section will be signed "over" by each maintainer, using an OIDC email address as the identifier, along with a signature and fulcio signing certificate populated. Each value is then stored in the `signatures` section.

Commands used :

1. To specify the list of maintainers, threshold and out file :

```
cosign policy init --namespace ghcr.io/jyotsna-penumaka/sigstore-kubecon --maintainers
lsturman@redhat.com,jpenumak@redhat.com,jyotsnap@bu.edu --threshold 2 --out root
```

2. To sign the policy:

```
cosign policy sign --namespace ghcr.io/jyotsna-penumaka/sigstore-kubecon --out signed_root
```

Generated outfile (signed_root):

```
{
    "signatures": [
        {
                "cert":
"LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNwakNDQWl1Z0F3SUJBZ0lUZDVVSTdRZnRjMHgwMEd1U25rVWp6TmFiaWpBS0J
nZ3Foa2pPUFFRREF6QXEKTVJVd0V3WURWUVFLRXd4emFXZHpkRzl5WlM1a1pYWXhFVEFQQmdOVkJBTVRDSE5wWjNOMGIzSmxNQjRYRFR
JeApNVEV4TnpFNU16WXhPVm9YRFRJeE1URXhOekU1TlRZeE9Gb3dBREJaTUJNR0J5cUdTTTQ5QWdFR0NDcUdTTTQ5CkF3RUhBMElBQlB
rU0ZuYWxBaEtuT2FRQlIyTkQ2SUJoUUFRV1FwMklvaThaRnZyTkMwVGVCdHFkRmV0NG9QNksKTWRvK09hU1pUM0poUTNnZFZoK09NM3V
LbmZydGQ4T2pnZ0ZZTUlJQlZEQU9CZ05WSFE4QkFmOEVCQU1DQjRBdwpFd1lEVlIwbEJBd3dDZ1lJS3dZQkJRVUhBd013REFZRFZSMFR
BUUgvQkFJd0FEQWRCZ05WSFE0RUZnUVVIRGtqCkxEMXVMRW4wR21GM0dlaDNXZjlwKzU4d0h3WURWUjBqQkJnd0ZvQVV5TVVkQUVHYUp
Da3lVU1RyRGE1SzdVb0cKMCt3d2dZMEdDQ3NHQVFVRkJ3RUJCSUdBTUg0d2ZBWUlLd1lCQlFVSE1BS0djR2gwZEhBNkx5OXdjbWwyWVh
SbApZMkV0WTI5dWRHVnVkQzAyTUROObVpUZGxOeTB3TURBd0xUSXlNamN0WW1ZM05TMW1OR1kxWlRnd1pESTVOVFF1CmMzUnZjbUZuWlM
```

```
1bmIyOW5iR1ZoY0dsekxtTnZiUzlqWVRNMllURmxPVFl5TkRKaU9XWmpZakUwTmk5allTNWoKY25Rd0lRWURWUjBSQVFIL0JCY3dGWUV
UYW5CbGJuVnRZV3RBY21Wa2FHRjBMbU52YlRBc0Jnb3JCZ0VFQVlPLwpNQUVCQkI1b2RIUndjem92TDJkcGRHaDFZaTVqYjIwdmJHOW5
hVzR2YjJGMWRHZ3dDZ11lJS29aSXpqMEVBd01ECmFRQXdaZ0l4QVB2NEJoU1BlUlV1T3drRFBuZkVWZHM1OElWUnBIazJDbysrSG16UUN
FZ3BxQkVrU0F5dUZuc1gKK0FaL042a1cwUUl4QUtiZGhIcVRWQVJORXhiVHY2YURCRENhelM4VGRqOHk1YkVtdG5MNmtGL2hON3BhcDd
hYgp3U2ZNdzNRQzFJQWllUT09Ci0tLS0tRU5EIENFUlRJRklDQVRFLS0tLS0K",
                "keyid": "db379d3746fe29d97f8e2dee8381d535a4b579d3ecf619613738cf138dadec0a",
                "sig":
"MEUCIQCyDrVdm84zwNV875OSHHPGKEGV0KL22+52c9CtowPn4gIgejx358uwTPaSuzX04SaCQc+iIsSh4svYQhtxwgFEo8A="
        }
    ],
    "signed": {
        "_type": "root",
        "consistent_snapshot": true,
        "expires": "2022-02-17T19:34:53Z",
        "keys": {
                "0dbdcea45bc3fa9091551690b89caa8bc322546b72fc6c766ccfa2be60547de6": {
                        "keyid_hash_algorithms": [
                                "sha256",
                                "sha512"
                        ],
                        "keytype": "sigstore-oidc",
                        "keyval": {
                                "identity": "jyotsnap@bu.edu",
                                "issuer": ""
                        },
                        "scheme": "https://fulcio.sigstore.dev"
                },
                "db379d3746fe29d97f8e2dee8381d535a4b579d3ecf619613738cf138dadec0a": {
                        "keyid_hash_algorithms": [
                                "sha256",
                                "sha512"
                        ],
                        "keytype": "sigstore-oidc",
                        "keyval": {
                                "identity": "jpenumak@redhat.com",
                                "issuer": ""
                        },
                        "scheme": "https://fulcio.sigstore.dev"
                },
                "e71beb853fb177ecd4248f1fe8c6e7c31476b8ff00842d53ecfff9332b7c70be": {
                        "keyid_hash_algorithms": [
                                "sha256",
                                "sha512"
                        ],
                        "keytype": "sigstore-oidc",
                        "keyval": {
                                "identity": "lsturman@redhat.com",
                                "issuer": ""
                        },
                        "scheme": "https://fulcio.sigstore.dev"
                }
        },
        "namespace": "ghcr.io/jyotsna-penumaka/sigstore-kubecon",
        "roles": {
                "root": {
                        "keyids": [
                                "e71beb853fb177ecd4248f1fe8c6e7c31476b8ff00842d53ecfff9332b7c70be",
                                "db379d3746fe29d97f8e2dee8381d535a4b579d3ecf619613738cf138dadec0a",
                                "0dbdcea45bc3fa9091551690b89caa8bc322546b72fc6c766ccfa2be60547de6"
```

```
                        ],
                        "threshold": 2
                }
        },
        "spec_version": "1.0",
        "version": 1
    }
}
```

- note: A policy could also just be a single maintainer (albeit with less security consensus)

The `root` file we then be pushed to `${registry_namespace}/root-policy`

For example `ghcr.io/lukehinds/myproject/root-policy`

All signing events for `root` will also be sent to rekor for inclusion.

Policy creation and signing will be utilized within sigstore/cosign (client implementations will be within a new code base ‘sget’)

Possible attack vector of root policy compromise

Should there be a compromise of a user's registry account credentials, an attacker could replace the entire root policy and generate their own signed & tampered scripts. It is therefore vital that we provide (at least at some point in the foreseeable future) a mechanism for a maintainer(s) (or any interested parties) to monitor rekor for events that contain their registry namespace and/or OIDC email. `sget` pull operations **have to fail / reject**, when there is no rekor entry found.

Another option is for the `sget` cli to create its own `XDG_CONFIG_HOME` cache, so that if a policy ownership deviation occurs (above the threshold of maintainers) a failure / warning occurs modelled after the ssh host tampering banner. This could be implemented with a `--strict` flag, that bails on any policy change. This may result in CI failure, but that may be a preferred outcome for the paranoid. Having said that, with CI nodes being ephemeral, this makes keeping a historic local cache a challenge. Also users are most often going to download scripts and run once.

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE POLICY IDENTIFICATION HAS CHANGED!   @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
....
```

root policy signing

The initial policy can be created as follows:

```
cosign policy init --namespace <project_namespace> --maintainers {email_addresses}
--threshold <int> --expires <int>(days) --out <filename>
```

e.g.

```
cosign policy init --namespace ghcr.io/jyotsna-penumaka/sigstore-kubecon
--maintainers lsturman@redhat.com,jpenumak@redhat.com,jyotsnap@bu.edu --threshold 2
--out root
```

An unsigned policy will then be produced as `root`:
Note : using root.json causes issues after uploading the file to GitHub, because of the "."


```
{
   "signatures": null,
   "signed": {
       "_type": "root",
       "consistent_snapshot": true,
       "expires": "2022-02-17T19:34:53Z",
       "keys": {
           "0dbdcea45bc3fa9091551690b89caa8bc322546b72fc6c766ccfa2be60547de6": {
               "keyid_hash_algorithms": [
                   "sha256",
                   "sha512"
               ],
               "keytype": "sigstore-oidc",
               "keyval": {
                   "identity": "jyotsnap@bu.edu",
                   "issuer": ""
               },
               "scheme": "https://fulcio.sigstore.dev"
           },
```

```
            "db379d3746fe29d97f8e2dee8381d535a4b579d3ecf619613738cf138dadec0a": {
                "keyid_hash_algorithms": [
                    "sha256",
                    "sha512"
                ],
                "keytype": "sigstore-oidc",
                "keyval": {
                    "identity": "jpenumak@redhat.com",
                    "issuer": ""
                },
                "scheme": "https://fulcio.sigstore.dev"
            },
            "e71beb853fb177ecd4248f1fe8c6e7c31476b8ff00842d53ecfff9332b7c70be": {
                "keyid_hash_algorithms": [
                    "sha256",
                    "sha512"
                ],
                "keytype": "sigstore-oidc",
                "keyval": {
                    "identity": "lsturman@redhat.com",
                    "issuer": ""
                },
                "scheme": "https://fulcio.sigstore.dev"
            }
        },
        "namespace": "ghcr.io/jyotsna-penumaka/sigstore-kubecon",
        "roles": {
            "root": {
                "keyids": [

"e71beb853fb177ecd4248f1fe8c6e7c31476b8ff00842d53ecfff9332b7c70be",

"db379d3746fe29d97f8e2dee8381d535a4b579d3ecf619613738cf138dadec0a",

"0dbdcea45bc3fa9091551690b89caa8bc322546b72fc6c766ccfa2be60547de6"
                ],
                "threshold": 2
            }
        },
        "spec_version": "1.0",
        "version": 1
    }
}
```


The initial user then pushes this file (payload) to a registry and signs the root policy

```
cosign policy sign --namespace ghcr.io/jyotsna-penumaka/sigstore-kubecon
--out signed_root
```

```
Generating ephemeral keys...
Retrieving signed certificate...
Your browser will now be opened to:
https://oauth2.sigstore.dev/auth/auth?access_type=online&client_id=sigstore
&code_challenge=tAcmxYLDnMY9w5zb13vASuPO6q0tnMCLlzDB31LjG1w&code_challenge_
method=S256&nonce=213li5tgBZ7DDmNeP83kDZ2ilm6&redirect_uri=http%3A%2F%2Floc
alhost%3A34189%2Fauth%2Fcallback&response_type=code&scope=openid+email&stat
e=213li3gxVVRKkcno8Goprj8lGPV
Successfully verified SCT...
Uploading file from [signed_root] to
[ghcr.io/jyotsna-penumaka/sigstore-kubecon/root:latest] with media type
[text/plain]
File [signed_root] is available directly at
[ghcr.io/v2/jyotsna-penumaka/sigstore-kubecon/root/blobs/sha256:8ac4028bb39
42db45bfcc858de9ce11a4a91cbe551bdbd18b2469d3f7aa75f40]
Uploaded image to:
ghcr.io/jyotsna-penumaka/sigstore-kubecon/root@sha256:89f95b55f79bf62e160b9
045702f8a7655c407314246dc60887909ab6519d771
```

This results in a first `signed` entry being generated under the signatures section and stored at uri of `ghcr.io/jyotsna-penumaka/sigstore-kubecon/root:latest`


```
{
   "signatures": [
      {
          "cert":
"LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNwakNDQWl1Z0F3SUJBZ0lUZDVVSTdRZnRjMHgwMEd1U25rVWp
6TmFiaWpBS0JnZ3Foa2pPUFFRREF6QXEKTVJVd0V3WURWUVFLRXd4emFXZHpkRzl5WlM1a1pYWXhFVEFQQmdOVkJBTVR
DSE5wWjNOMGIzSmxNQjRYRFRJeApNVEV4TnpFNU16WXhPVm9YRFRJeE1URXhOekU1TlRZeE9Gb3dBREJaTUJNR0J5cUd
TTTQ5QWdFR0NDcUdTTTQ5CkF3RUhBMElBQlBrU0ZuYWxBaEtuT2FRQlIyTkQ2SUJoUUFRV1FwMklvaThaRnZyTkMwVGV
CdHFkRmV0NG9QNksKTWRvK09hU1pUM0poUTNnZFZoK09NM3VLbmZydGQ4T2pnZ0ZZTUlJQlZEQU9CZ05WSFE4QkFmOEV
CQU1DQjRBdwpFd1lEVlIwbEJBd3dDZ1lJS3dZQkJRVUhBd013REFZRFZSMFRBUUgvQkFJd0FEQWRCZ05WSFE0RUZnUVV
IRGtqCkxEMXVMRW4wR21GM0dlaDNXZjlwKzU4d0h3WURWUjBqQkJnd0ZvQVV5TVVkQUVHYUpDa3lVU1RyRGE1SzdVb0c
KMCt3d2dZMEdDQ3NHQVFVRkJ3RUJCSUdBTUg0d2ZBWUlLd1lCQlFVSE1BS0djR2gwZEhBNkx5OXdjbWwyWVhSbApZMkV
0WTI5dWRHVnVkQzAyTUROObVpUZGxOeTB3TURBd0xUSXlNamN0WW1ZM05TMW1OR1kxWlRnd1pESTVOVFF1CmMzUnZjbUZ
uWlM1bmIyOW5iR1ZoY0dsekxtTnZiUzlqWVRNMllURmxPVFl5TkRKaU9XWmpZakUwTmk5allTNWoKY25Rd0lRWURWUjB
SQVFIL0JCY3dGWUVUYW5CbGJuVnRZV3RBY21Wa2FHRjBMbU52YlRBc0Jnb3JCZ0VFQVlPLwpNQUVCQkI1b2RIUndjem9
2TDJkcGRHaDFZaTVqYjIwdmJHOW5hVzR2YjJGMWRHZ3dDZ1lJS29aSXpqMEVBd01ECmFRQXdaZ0l4QVB2NEJoU1BlUlV
1T3drRFBuZkVWZHM1OElWUnBIazJDbysrSG16UUNFZ3BxQkVrU0F5dUZuc1gKK0FaL042a1cwUUl4QUtiZGhIcVRWQVJ
ORXhiVHY2YURCRENhelM4VGRqOHk1YkVtdG5MNmtGL2hON3BhcDdhYgp3U2ZNdzNRQzFJQWllUT09Ci0tLS0tRU5EIEN
FUlRJRklDQVRFLS0tLS0K",
```

```
            "keyid": "db379d3746fe29d97f8e2dee8381d535a4b579d3ecf619613738cf138dadec0a",
            "sig":
"MEUCIQCyDrVdm84zwNV875OSHHPGKEGV0KL22+52c9CtowPn4gIgejx358uwTPaSuzX04SaCQc+iIsSh4svYQhtxwgF
Eo8A="
        }
    ],
    "signed": {
        "_type": "root",
        "consistent_snapshot": true,
        "expires": "2022-02-17T19:34:53Z",
        "keys": {
            "0dbdcea45bc3fa9091551690b89caa8bc322546b72fc6c766ccfa2be60547de6": {
                "keyid_hash_algorithms": [
                    "sha256",
                    "sha512"
                ],
                "keytype": "sigstore-oidc",
                "keyval": {
                    "identity": "jyotsnap@bu.edu",
                    "issuer": ""
                },
                "scheme": "https://fulcio.sigstore.dev"
            },
            "db379d3746fe29d97f8e2dee8381d535a4b579d3ecf619613738cf138dadec0a": {
                "keyid_hash_algorithms": [
                    "sha256",
                    "sha512"
                ],
                "keytype": "sigstore-oidc",
                "keyval": {
                    "identity": "jpenumak@redhat.com",
                    "issuer": ""
                },
                "scheme": "https://fulcio.sigstore.dev"
            },
            "e71beb853fb177ecd4248f1fe8c6e7c31476b8ff00842d53ecfff9332b7c70be": {
                "keyid_hash_algorithms": [
                    "sha256",
                    "sha512"
                ],
                "keytype": "sigstore-oidc",
                "keyval": {
                    "identity": "lsturman@redhat.com",
                    "issuer": ""
                },
                "scheme": "https://fulcio.sigstore.dev"
            }
        },
        "namespace": "ghcr.io/jyotsna-penumaka/sigstore-kubecon",
        "roles": {
            "root": {
                "keyids": [
```

```
                    "e71beb853fb177ecd4248f1fe8c6e7c31476b8ff00842d53ecfff9332b7c70be",
                    "db379d3746fe29d97f8e2dee8381d535a4b579d3ecf619613738cf138dadec0a",
                    "0dbdcea45bc3fa9091551690b89caa8bc322546b72fc6c766ccfa2be60547de6"
                ],
                "threshold": 2
            }
        },
        "spec_version": "1.0",
        "version": 1
    }
}
```


After this further maintainers may also sign over the body to at least the threshold

```
cosign policy sign --namespace ghcr.io/jyotsna-penumaka/sigstore-kubecon
```

This in turn pulls down the existing policy and signs over the body and pushes a new revision to the registry and to rekor. Each time a signing occurs, the latest becomes the active policy.


```
{
    "signatures": [
        {
            "cert":
"LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNwakNDQWl1Z0F3SUJBZ0lUZDVVSTdRZnRjMHgwMEd1U25rVWp
6TmFiaWpBS0JnZ3Foa2pPUFFRREF6QXEKTVJVd0V3WURWUVFLRXd4emFXZHpkRzl5WlM1a1pYWXhFVEFQQmdOVkJBTVR
DSE5wWjNOMGIzSmxNQjRYRFRJeApNVEV4TnpFNU16WXhPVm9YRFRJeE1URXhOekU1TlRZeE9Gb3dBREJaTUJNR0J5cUd
TTTQ5QWdFR0NDcUdTTTQ5CkF3RUhBMElBQlBrU0ZuYWxBaEtuT2FRQlIyTkQ2SUJoUUFRV1FwMklvaThaRnZyTkMwVGV
CdHFkRmV0NG9QNksKTWRvK09hU1pUM0poUTNnZFZoK09NM3VLbmZydGQ4T2pnZ0ZZTUlJQ1ZEQU9CZ05WSFE4QkFmOEV
CQU1DQjRBdwpFd1lEVlIwbEJBd3dDZ1lJS3dZQkJRVUhBd013REFZRFZSMFRBUUgvQkFJd0FEQWRCZ05WSFE0RUZnUVV
IRGtqCkxEMXVMRW4wR21GM0dlaDNXZjlwKzU4d0h3WURWUjBqQkJnd0ZvQVV5TVVkQUVHYUpDa3lVU1RyRGE1SzdVb0c
KMCt3d2dZMEdDQ3NHQVFVRkJ3RUJCSUdBTUg0d2ZBWUlLd1lCQlFVSE1BS0djR2gwZEhBNkx5OXdjbWwyWVhSbApZMkV
0WTI5dWRHVnVkQzAyTUROObVpUZGxOeTB3TURBd0xUSXlNamN0WW1ZM05TMW1OR1kxWlRnd1pESTVOVFF1CmMzUnZjbUZ
uWlM1bmIyOW5iR1ZoY0dsekxtTnZiUzlqWVRNMllURmxPVFl5TkRKaU9XWmpZakUwTmk5allTNWoKY25Rd0lRWURWUjB
SQVFIL0JCY3dGWUVUYW5CbGJuVnRZV3RBY21Wa2FHRjBMbU52YlRBc0Jnb3JCZ0VFQVlPLwpNQUVCQkI1b2RIUndjem9
2TDJkcGRHaDFZaTVqYjIwdmJHOW5hVzR2YjJGMWRHZ3dDZ1lJS29aSXpqMEVBd01ECmFRQXdaZ0l4QVB2NEJoU1BlUlV
1T3drRFBuZkVWZHM1OElWUnBIazJDbysrSG16UUNFZ3BxQkVrU0F5dUZuc1gKK0FaL042a1cwUUl4QUtiZGhIcVRWQVJ
ORXhiVHY2YURCRENhelM4VGRqOHk1YkVtdG5MNmtGL2hON3BhcDdhYgp3U2ZNdzNRQzFJQWllUT09Ci0tLS0tRU5EIEN
FUlRJRklDQVRFLS0tLS0K",
            "keyid": "db379d3746fe29d97f8e2dee8381d535a4b579d3ecf619613738cf138dadec0a",
            "sig":
"MEUCIQCyDrVdm84zwNV875OSHHPGKEGV0KL22+52c9CtowPn4gIgejx358uwTPaSuzX04SaCQc+iIsSh4svYQhtxwgF
Eo8A="
        },
        {
            "cert":
"LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNuakNDQWlXZ0F3SUJBZ0lVQU5QWW56Rzd2d0tySCt3aXh1RzB
hWHpUMjBRd0NnWUlLb1pJemowRUF3TXcKS2pFVk1CTUdBMVVFQ2hNTWMybG5jM1J2Y21VdVpHVjJNUkV3RHdZRFZRUUR
```

```
Fd2h6YVdkemRHOXlaVEFlRncweQpNVEV4TVRjeU1EQXpORGxhRncweU1URXhNVGN5TURJek5EaGFNQUF3V1RBVEJnY3F
oa2pPUFFJQkJnZ3Foa2pPClBRTUJCd05DQUFRUHFRTDMyN2UrT0FLYTJEVWRnN2tPTDdVaGFQZEJtY1ZjUUJvWFZaMHp
BTHg2SytpbHlQZEkKOFdEb05GaFZreFdIa0NmSkJWeGlrYnNaODMzQkZLRGNvNElCVVRDQ0FVMHdEZ1lEVlIwUEFRSC9
CQVFEQWdlQQpNQk1HQTFVZEpRUU1NQW9HQ0NzR0FRVUZCd01ETUF3R0ExVWRFd0VCL3dRQ01BQXdIUVlEVlIwT0JCWUV
GQVYyCmY1THlURE1iZFBRVW5lSWlqd1VkRmRMc01COEdBMVVkSXdRWU1CYUFGTWpGSFFCQm1pUXBNbEVrNncydVN1MUs
KQnRQc01JR05CZ2dyQmdFRkJRY0JBUVNCZ0RCK01Id0dDQ3NHQVFVRkJ6QUNobkJvZEhSd09pOHZjSEpwZG1GMApaV05
oTFdOdmJuUmxiblF0TmpBelptVTNaVGN0TURBd01DMHlNakkzTFdKbU56VXRaalJtTldVNE1HUXlPVFUwCkxuTjBiM0p
oWjJVdVoyOXZaMnhsWVhCcGN5NWpiMjB2WTJFek5tRXhaVGsyTWpReVlqbG1ZMkl4TkRZdlkyRXUKWTNKME1CMEdBMVV
kRVFFQi93UVRNQkdCRDJwNWIzUnpibUZ3UUdKMUxtVmtkVEFwQmdvckJnRUVBWU8vTUFFQgpCQnRvZEhSd2N6b3ZMMkZ
qWTI5MWJuUnpMbWR2YjJkc1pTNWpiMjB3Q2dZSUtvWkl6ajBFQXdNRFp3QXdaQUl3ClMybVVZUXVNVHFFeVpPTWJnZGx
KR08xVHArZ3RicW5Wd0hobDg1bWV5cTVJUU4wZVcvZDQvblFsdk1NN0ZUcnYKQWpCQkhMMXFnM3RSUkpLcWsva2Q4YjF
RMzNHOWNmUjJrQTVOUWM2dUQvUnlrWEt2L2hPWk5Ob1Ztc21Vb05UQwpIV0k9Ci0tLS0tRU5EIENFUlRJRklDQVRFLS0
tLS0K",
            "keyid": "0dbdcea45bc3fa9091551690b89caa8bc322546b72fc6c766ccfa2be60547de6",
            "sig":
"MEQCIHeA07VLfdF+sHTgEpMy5YKvfg7DKCp+eoZ55nree0gyAiAsAzqe2zXqGfAFHWmucOnii9M+Z3u/PcD2NNiWCXy
B3A=="
        }
    ],
    "signed": {
        "_type": "root",
        "consistent_snapshot": true,
        "expires": "2022-02-17T19:34:53Z",
        "keys": {
            "0dbdcea45bc3fa9091551690b89caa8bc322546b72fc6c766ccfa2be60547de6": {
                "keyid_hash_algorithms": [
                    "sha256",
                    "sha512"
                ],
                "keytype": "sigstore-oidc",
                "keyval": {
                    "identity": "jyotsnap@bu.edu",
                    "issuer": ""
                },
                "scheme": "https://fulcio.sigstore.dev"
            },
            "db379d3746fe29d97f8e2dee8381d535a4b579d3ecf619613738cf138dadec0a": {
                "keyid_hash_algorithms": [
                    "sha256",
                    "sha512"
                ],
                "keytype": "sigstore-oidc",
                "keyval": {
                    "identity": "jpenumak@redhat.com",
                    "issuer": ""
                },
                "scheme": "https://fulcio.sigstore.dev"
            },
            "e71beb853fb177ecd4248f1fe8c6e7c31476b8ff00842d53ecfff9332b7c70be": {
                "keyid_hash_algorithms": [
                    "sha256",
                    "sha512"
```

```
                ],
                "keytype": "sigstore-oidc",
                "keyval": {
                    "identity": "lsturman@redhat.com",
                    "issuer": ""
                },
                "scheme": "https://fulcio.sigstore.dev"
            }
        },
        "namespace": "ghcr.io/jyotsna-penumaka/sigstore-kubecon",
        "roles": {
            "root": {
                "keyids": [
                    "e71beb853fb177ecd4248f1fe8c6e7c31476b8ff00842d53ecfff9332b7c70be",
                    "db379d3746fe29d97f8e2dee8381d535a4b579d3ecf619613738cf138dadec0a",
                    "0dbdcea45bc3fa9091551690b89caa8bc322546b72fc6c766ccfa2be60547de6"
                ],
                "threshold": 2
            }
        },
        "spec_version": "1.0",
        "version": 1
    }
}
```

signature manifest

The following is proposed structure for a signature manifest generated for a signed script (script)

- `body` contains annotations with a security context.
    - `registry_namespace`: fixes the manifest to a registry namespace
    - `layer_digest`: as it says
    - `sha256`: A SHA 256 digest of the script
    - `date`: timestamp of when the script was generated
- `signed`: maintainers signing the manifest's `body` section

```
{
"signed": {
  "registry_namespace": "ghcr.io/lukehinds/myscript.sh",
  "layer_digest": "d93jd0si97cf3baf3507d338a7637d927863bd9c55ed21cb7f11ee8f734dde",
  "sha256": "12a71af6ef97cf3baf3507d338a7637d927863bd9c55ed21cb7f11ee8f73631e",
  "date": "2021-06-25T00:00:00.000Z",
},
"signatures": [
  {
    "email": "bill@example.com",
    "fuclio_cert": "-----BEGIN CERTIFICATE----- \nMIIIPDCCBySgAwIBAgIQB2XGTnTlkda...",
    "signature": "a337d6375fedd2eabfcd6c2ef6c8a9c3bb85dc5a85.."
  },
  {
    "email": "jenny@email.com",
    "fuclio_cert": "-----BEGIN CERTIFICATE----- \nM1SISPDCCBySgAwIBAgIQB2XGTnTlker...",
    "signature": "23e34rreerwedd2eabfcd6c2ef6c8a9c3bb85dc5a85.."
  }
 ]
}
```

Script signing process

`cosign sign` works with the premise of there being no current script / signing manifest (**initial signing**) **or** an existing script / signing manifest (**subsequent signing**). If an existing signature / script is present, then signatures are stacked onto the existing `signed` section.

Initial signing

```
$ cosign sign-blob --key <key path>|<kms uri>
$ cosign upload blob -f <blob ref> <image uri>
```

If no `--file` flag is passed to the sign operation it will attempt to sign the existing entry (see **subsequent signing).** If the `--file` flag is used, and an entry (script) already exists, the user will be warned and presented with a prompt to proceed. If the user selects `'Y'` to the prompt, the entire signature manifest will be refreshed (this of course will only happen for existing maintainers).

The following flow will occur (*italics* denote existing standard functionality)

1. *OpenID session grant request*
2. *User provides an OIDC grant or if no user is present, the system pivots to device flow.*
3. *Ephemeral key pair is created*
4. *Using IDToken from step 1, public key is submitted to fulcio*
5. *Signing certificate is returned.*
6. The script is first pushed to the registry and the `layer_digest` is captured.
7. A json payload is crafted with the body that contains the `registry_namespace`, the `sha256`, `layer_digest`, `date`
8. The above payload body is signed and the signing event is captured under `signed` section
9. The payload (signature manifest) is pushed to the registry
10. Signature, `body` digest and x509 signing cert are pushed to rekor

subsequent signing

If no `--file` flag is parsed the sign operation will sign the existing script in annotation `dev.sigstore.sget/signature_manifest`.

The following flow will then play out (*italics* denote existing standard functionality)

1. *OpenID session proceed*
2. *User provides an OIDC grant or if no user is present, the system pivots to device flow.*
3. The script manifest will be retrieved, along with the script digest
4. *Ephemeral key pair is created*
5. *Using IDToken from step 1, public key is submitted to fulcio with challenge*
6. *Signing certificate is returned.*
7. script is signed
8. An updated signature manifest is pushed to the registry namespace annotation `dev.sigstore.sget/signature_manifest`
9. signature / script digest and x509 signing cert are pushed to rekor

SGET user operations

`sget` will be a new code base for user operations only. The bulk of sget's operations will be validation of the script signing materials against the root policy. If this validation passes, the script can be executed.

subflags

- `--noexec` : retrieve the script, validate signing but ****do not execute"***
- `--out` if coupled with `--noexec` a posix path can be provided to download the script

Validation

The sget pull operation will follow this sequence of events

1. The signing manifest, root policy and script are retrieved from the provided registry namespace
2. The root policy manifest is processed and validated:
    1. All signatures under the `signed` section of the root policy is validated against the `body`
    2. All fulcio certs are validated as chained to the fulcio root.
    3. The `expires` section is validated as being correct (not in the past)

4. The policy manifest is confirmed as being in rekor with inclusion entries for all maintainers (if any are missing, we fail)
5. If #1 to #4 pass validation, all email addresses, the registry name space and the threshold are stored in memory for signature manifest validation. If any fail, we exit with a failure code

1. The signature manifest is processed validated
2. Each ODIC email address is validated as being present in the root policy
3. The amount of ODIC email addresses in the signing manifest is equal or greater to the root policy `threshold`
4. The `registry_namespace` is validated as being identical to the `registry_namespace` set within the root policy
5. The signature for each maintainer is verified as correctly signing the `sha256` digest of the script
6. The fulcio certs are attested as having a chain to the sigstore fulcio root CA
7. Rekor entries are present for every maintainer's signing event.
8. Execute script (when all above validations are passed and --noexec flag is false

Packaging

For greater adoption, a sget needs to be widely packaged so it can easily be pulled in as a package within CI or developers machines. Packaging should be made available for as many Linux Distributions as possible, along with homebrew and other pkg management systems

- Fedora
- Debian
- Arch Linux
- Homebrew
- Windows(??)

Project onboarding

When sget has a good coverage of OS packaging availability, projects will be contacted for onboarding.

A list of projects utilizing internet hosted shell scripts for pipe execution have been gathered: https://docs.google.com/spreadsheets/d/1ZnrVx8Bz-0cCjoX0JgIuf8rb6QwPTkII2Wx9ajYoGtA/edit?usp=sharing

Each of these projects will be contacted and sigstore community members will assist them to onboard projects. Each project will be encouraged to share their experience with [sigstore/friends]