TLS

certificate

Client ⟶ Server

Encryption keys

Symmetric Encryption → same key to encrypt & decrypt

Asymmetric Encryption

Data encrypted with public lock/key. can be decrypted only using private key.

SSH Encryption

$ ssh keygen

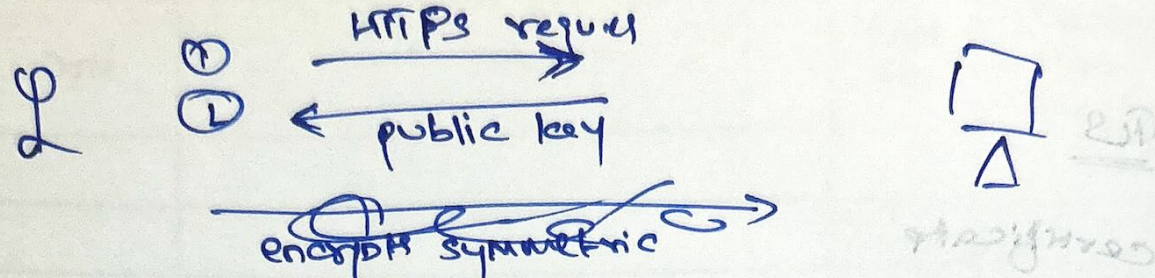
id_rsa id_rsa.pub

↳ send this to server.

for HTTPS,

openssl genrsa -out

my-bank.key my-bank.pem



③ Encrypt symmetric key with public key provided by server

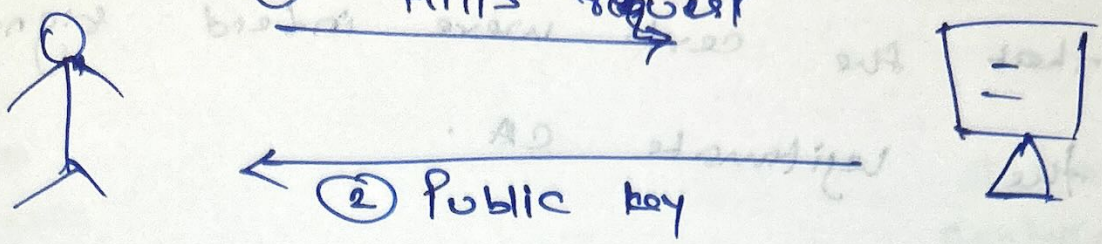
④ Server uses private key to decrypt & extract the symmetric key

Same symmetric key now used to encrypt & decrypt info.

Certificate with key

Self-signed certificate

Certificate Authority



③ Encrypt symmetric key using public key provided by server & send it over

④ Server uses private key to decrypt & extract the symmetric key

Same symmetric key used to encrypt & decrypt info

Certificate signing request (CSR)

Validate information

Sign & send certificate

CA use their private key to sign the certificate

The public keys of all the CA's are built into the browser

Browser uses these public keys to verify that the certs were indeed signed by the legitimate CA.

To access internal resources, that are not exposed to internet, we can use private CA. Example: Symantec Private CA

Server

You can have the public key of your internal CA server installed on all employee browsers.

This is all called PKI (Public key Infrastructure)

Extensions for Public key → .crt, .pem

Private key → .key, -key.pem

Securing K8s cluster with TLS certificates

Client certificates (client-side)

Server certificates (server side)

root certificates (CA)

1) Server certificates for servers
2) Client certs for clients

Server certs

- kube-apiserver apiserver.crt apiserver.key
- etcd server etcdserver.crt etcdserver.key
- kubelet server

Client certs / servers who access above services

- admin admin.crt admin.key
- kube-scheduler
- kube-controller
- kube-proxy

kube-apiserver (client) → etcd-server (server)

We can have more than one CA per server.

Generating TLS certificates

Openssl tool to generate certs.

$ openssl genrsa -out ca.key 2048

Generate a key private

$ openssl req -new -key ca.key -subj "/CN=KUBERNETES-CA" -out ca.csr

Certificate signing request

$ openssl -x509 -req -in ca.csr -signkey ca.key -out ca.crt

Sign certificate

Here, self-signed

ca.csr: contains all details, but without signature

CN: Common Name

Here, self-signed by the CA using its own private key that it generated in first step.

Going forward, for all other certificates, we will use the ca key pair to sign them.

The CA now has its private key & root certificate file.

Now, you can generate client certificates.

1. Admin user
   - Generate keys

$ openssl genrsa -out admin.key 2048

admin.key

   - Certificate signing request

$ openssl req -new -key admin.key -subj "/CN=kube-admin" -out admin.csr

admin.csr

   - Sign certificate

$ openssl x509 -req -in admin.csr -CA ca.crt -CAkey ca.key -out admin.crt

admin.crt

You're signing your certificate with the CA key pair. That make this a valid certificate working the cluster.

admin.crt is the certificate that the admin user will use to authenticate to k8s cluster.

Similarly create client certificates for all clients.

You can now use these certificates to authenticate to api-server

```
$ curl https://kube-apiserver:6443/api/v1/pods
  --key admin.key --cert admin.crt
  --cacert ca.crt
```

Certificate data can be moved to kube-config.yaml

In k8s, for these various components to verify each other, they all need a copy of CA root certificate. So, whenever you configure a server or a client with certs, you will need to specify the CA root cert as well.

Certificates must similarly be generated for services as well - kube-apiserver, etcd-server, kubelet-servers, etc.

Location of certificate is passed into the kube-apis server's executable or services configuration file. Includes CA file, API server certificates under tls-cert options, client certs used by kube-api server to connect to etcd, etc.

Create certs for kubelet on each node. Use them in kubelet config file. Each node needs it's own kubelet config file.

Viewing certificate details

If cluster was set-up using kube-adm,

```
$ cat /etc/kubernetes/manifests/kube-apiserver.yaml
  spec:
    Containers:
    - command:
        - --client-ca-file=/etc/...
        - --etcd-ca-file=...
        - --tls-cert-file=...
```

Take each certificate & look inside it to find more details.

```
$ openssl x509 -in /etc/kubernetes/pki/api-server.crt
      -text -noout
  Certificate:
    Data:
      :
      :
      Subject: CN=kube-apiserver
```

Inspecting service logs

$ journalctl -u etcd.service -l

If kube-adm

$ kubectl logs etcd-master

Certificate API

The CA is really just a pair of key & certificate files we have generated. So, wherever these files are saved, that becomes our CA server.

K8s has a built-in certificates API

When the admin recieves a certificate signing request, instead of logging into the master node & signing the cert himself, he creates an API object:

1) Create CertificateSigningRequest object
2) Such requests can be seen by administrators and reviewed.

3) Approve request

4) Certs extracted & shared with user