

HAUTE ÉCOLE DU PAYSAGE, D'INGÉNIERIE  
ET D'ARCHITECTURE DE GENÈVE



ATELIERS EN SÉCURITÉ

SCÉNARIO D'INFRASTRUCTURE À CLEF PUBLIQUE

---

# Créer une autorité de certification avec Vault et Kubernetes

---

RAPPORT INTERMÉDIAIRE

*Auteurs:*

Flavio MORRONE

Elio MARCONI

Léo MUFF

June 6, 2024

## Contents

0.1	Introduction . . . . .	2
0.1.1	Idée principale . . . . .	2
0.1.2	Répartition des tâches . . . . .	2
0.1.3	Avancement . . . . .	2
0.2	Description de l'infrastructure à clef publique . . . . .	3
0.2.1	Chaîne de confiance . . . . .	3
0.2.2	Composants logiciels . . . . .	4
0.3	Mise en place de l'infrastructure : . . . . .	5
0.3.1	Installation Vault sur MicroK8s . . . . .	5
0.3.2	Installation d'un parser JSON . . . . .	7
0.3.3	Création de l'autorité de certification . . . . .	7
0.3.3.1	Création du CA racine: . . . . .	7
0.3.3.2	Création du CA intermédiaire : . . . . .	7
0.3.3.3	Création du rôle : . . . . .	8
0.3.3.4	Demander un certificat : . . . . .	8
0.3.4	Configuration de cert-manager . . . . .	8
0.3.4.1	Activation de cert-manager : . . . . .	8
0.3.4.2	Configurer l'accès entre Vault et Kubernetes . . . . .	8
0.3.4.3	Configurer l'émetteur de certificats . . . . .	9
0.3.5	Installer Nginx Ingress . . . . .	9
0.3.6	Mise en place de TLS sur un service . . . . .	10
0.3.6.1	Création du service . . . . .	10
0.3.6.2	Création du certificat du service : . . . . .	11
0.3.6.3	Configuration de l'accès au service . . . . .	12
0.3.6.4	Test de l'accès au service . . . . .	13
0.3.7	Mise à jour des certificats . . . . .	16
0.3.7.1	Suppression des certificats expirés . . . . .	16
0.3.7.2	Rotation du certificat racine . . . . .	16
0.3.7.3	Création d'un pont entre le nouveau certificat racine et l'ancien . . . . .	16
0.3.7.4	Remplacer l'émetteur par défaut . . . . .	16
0.3.7.5	Signer l'autorité intermédiaire avec notre nou- veau certificat racine . . . . .	17
0.3.8	Mise en place du service ACME . . . . .	17
0.3.8.1	terminal : . . . . .	17
0.3.8.2	GUI : . . . . .	17
0.3.9	Connecter Cert-manager au service ACME . . . . .	18
0.3.9.1	Ajout d'un émetteur . . . . .	18
0.4	Glossaire . . . . .	18
0.5	Références . . . . .	19

## 0.1 Introduction

Dans le cadre du cours “Ateliers en sécurité”, il nous a été demandé de créer une entreprise fictive qui utilise une infrastructure à clef publique, ou PKI (Public Key Infrastructure), pour sécuriser ses services. Ce document décrit le scénario choisi, l’organisation de notre entreprise fictive et les différents composants physiques et logiciels utilisés pour mettre en place notre infrastructure. Il décrit aussi la politique de certification utilisée pour celle-ci.

### 0.1.1 Idée principale

Nous avons choisi de concevoir une entreprise fournissant une version simplifiée des services proposés par une autorité de certification comme Let’s Encrypt. Notre infrastructure comprendra donc sa propre autorité de certification qui permettra d’émettre des certificats pour nos services ainsi que pour les services publiques de nos clients. La gestion de ces certificats pourra ensuite être automatisée grâce au protocole ACME (Automated Certificate Management Environment).

### 0.1.2 Répartition des tâches

- Recherches théoriques : Élio Marconi
- Mise en place de l’infrastructure : Flavio Morrone, Léo Muff
- Rédaction de la documentation : Léo Muff

### 0.1.3 Avancement

- ☒ Installation de Microk8s et Vault
- ☒ Configuration initiale de Vault
- ☒ Création de l’autorité racine et de l’autorité intermédiaire
- ☒ Configuration de Cert-manager
- ☒ Mise en place d’un service test en https
- ☒ Mise à jour des certificats
- ☒ Mise en place d’ACME
- ☐ Mise en place d’un service DNS pour valider les challenges ACME
- ☐ Mise en place d’un script pour permettre une utilisation simplifiée de notre service ACME.

## 0.2 Description de l'infrastructure à clef publique

La section ci-dessous décrit les composants et la logique interne de notre entreprise fictive, qui sera appelée **Chepia**.

### 0.2.1 Chaîne de confiance

Notre autorité racine émettra les certificats deux deux autorités intermédiaires : une pour émettre les certificats des services internes de notre entreprise, et une pour les certificats fourni avec ACME aux services externes de nos clients.

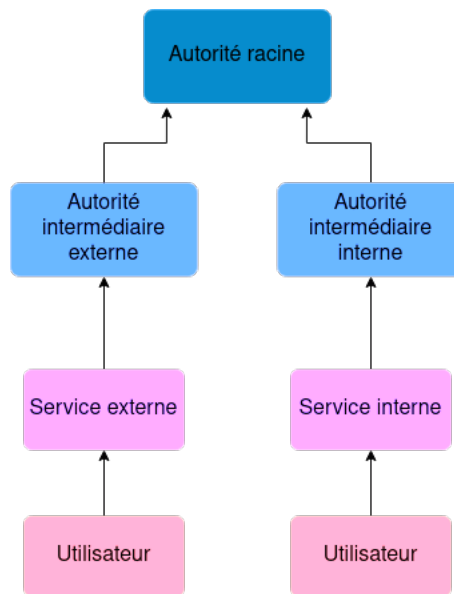


Figure 1: Chaîne de confiance de notre autorité de certification

### 0.2.2 Composants logiciels

- **MicroK8s** : Déploiement Kubernetes dans un seul package. Conçu pour le développement de services conteneurisés sur une machine locale.
- **Vault** : Outil de gestion de secrets qui propose de nombreuses fonctionnalités facilitant la mise en place d'une infrastructure à clé publique.
- **Cert-manager** : Outil de gestion de certificat pour Kubernetes. Son rôle sera de gérer le processus de certification pour nos services via notre gestionnaire de secrets Vault.
- **Nginx** : Serveur web. Utilisé pour tester le fonctionnement de notre autorité de certification.
- **ACME** (Automated Certificate Management Environment) : Protocole utilisé pour automatiser le processus de certification. Il nous permettra de créer un service similaire à celui de Let's Encrypt, et fournir des certificats aux services de nos clients.

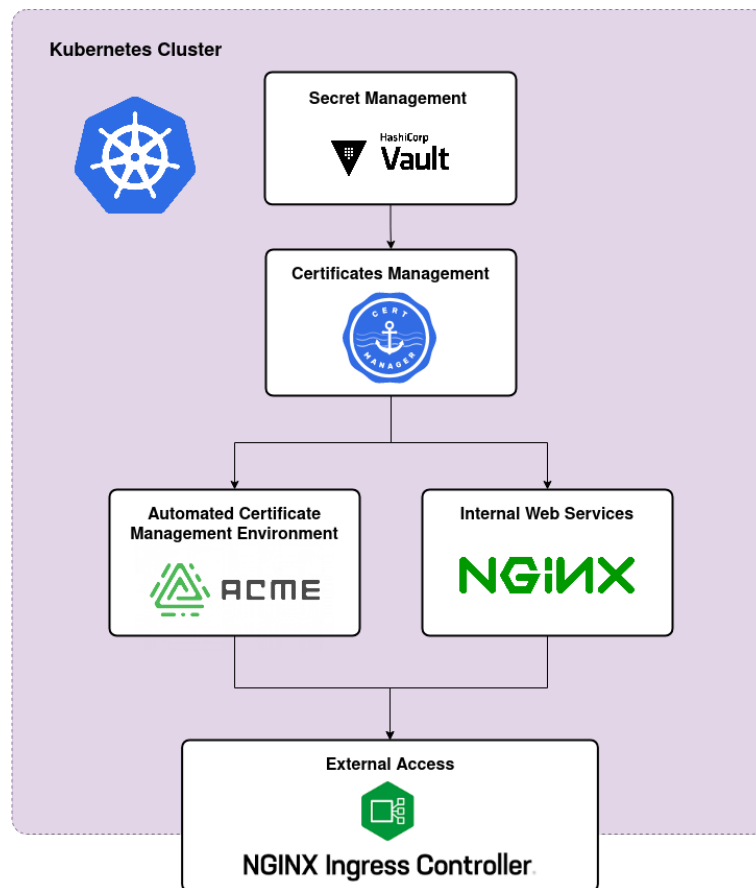


Figure 2: Pile logicielle de notre infrastructure

### 0.3 Mise en place de l'infrastructure :

→ Les commandes commençant par `microk8s` sont à exécuter dans le terminal de la machine hôte. Celles commençant par `vault` doivent être exécutées dans le container de Vault.

#### 0.3.1 Installation Vault sur MicroK8s

Les commandes suivantes permettent d'installer `microk8s` et d'y déployer une instance de Vault. Elles ont été testées sur un système Ubuntu 22.04, et peuvent varier selon le système utilisé.

```
sudo snap install microk8s --classic --channel=1.29
microk8s enable dns hostpath-storage dashboard
microk8s helm repo add hashicorp https://helm.releases.hashicorp.com
microk8s helm install vault hashicorp/vault
```

Une fois Vault déployé, le gestionnaire de secret doit être initialisé. Cette opération va créer une clef privée qui sera utilisée pour chiffrer les données de Vault. Cette clef est ensuite chiffrée par une autre clé aussi appelée *root key* qui est elle-même chiffrée par une autre clef appelée *unseal key*. La clé *unseal* est ensuite partagée en plusieurs parties selon le partage de secret de Shamir. Les différentes parties devront ensuite être réunies pour déverrouiller Vault.

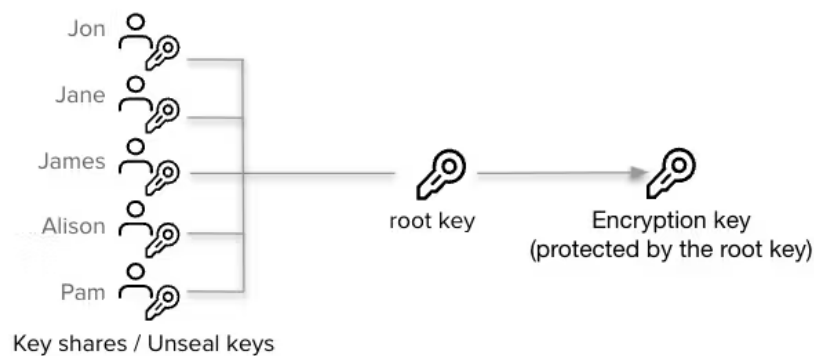


Figure 3: Secret partagé de Shamir

L'initialisation de Vault s'effectue avec la commande suivante :

```
microk8s kubectl exec -ti vault-0 -- vault operator init
```

→ Le nom du container `vault-0` peut être différent sur votre système. La commande `microk8s kubectl get pods -A` permet de vérifier cela.

Le résultat devrait ressembler à ceci :

```
tokens :
Unseal Key 1: 80pqLahtRotMHBFUVDlw4Ax3MU3aPfZAvFhRe7SFoW0l
Unseal Key 2: dBShI3QRLLoEmu9bT5BV/mq0yxPC8HqmiETwxpeuD7qc
Unseal Key 3: q+uRd7IB1GuejUz+RWoc+iuUuWxC3t2S53vD03pqlFEa
Unseal Key 4: 5/G/CndGOYuDR6X+/zMeMyK8BbswRuWpPexf0zswaNwt
Unseal Key 5: Oesp4xJ3xTElpgJm+bRfQ9Ia0cYdxiQZa3h9o3hQxK3c
```

Initial Root Token: `hvs.zWAYhaUkch0hfgfi18fduTv1`

Ces informations doivent être stockées de manière sécurisée, en utilisant un HSM (Hardware Security Module) par exemple. Une fois en possession de ces informations, on peut accéder aux fonctionnalités de Vault.

Pour accéder à l'interface graphique de Vault, on utilise la commande suivante :

```
microk8s kubectl port-forward vault-0 8200:8200
```

→ L'interface sera disponible à l'adresse `127.0.0.1:8200`


## Unseal Vault

**Vault is sealed**

Unseal Vault by entering portions of the unseal key. This can be done via multiple mechanisms on multiple computers. Once all portions are entered, the root key will be decrypted and Vault will unseal.

**Unseal Key Portion**

Unseal

2/3 keys provided 

[Seal/unseal documentation](#)

Figure 4: Interface graphique de Vault verrouillée

→ Pour déverrouiller Vault, il faut que 3 des 5 tokens créés soit renseignés (n'importe lesquels). On doit ensuite se logger avec un token utilisateur (root par défaut)

### 0.3.2 Installation d'un parser JSON

Vault est aussi utilisable en ligne de commande. Pour cela, il faut se connecter au container avec la commande suivante :

```
microk8s kubectl exec -ti vault-0 -- sh
```

Une fois connectés au container, on peut utiliser l'outil `vault` en ligne de commande. Comme cet outil génère principalement du JSON en sortie, il est nécessaire d'installer le parser JQ pour pouvoir réutiliser la sortie facilement.

Commandes :

```
cd vault
wget https://github.com/stedolan/jq/releases/download/jq-1.7.1/jq-linux64 -O jq
chmod +x jq
```

### 0.3.3 Création de l'autorité de certification

Une fois Vault prêt, on peut commencer à mettre en place notre PKI. Les étapes ci-dessous utilise la ligne de commande mais une correspondance en interface graphique est possible, voir la documentation de Vault pour cela.

#### 0.3.3.1 Création du CA racine:

Commandes :

```
microk8s kubectl exec -ti vault-0 -- sh
export VAULT_TOKEN=hvs.zWAYhaUkch0hfgfi18fduTvl
vault secrets enable pki
vault secrets tune -max-lease-ttl=336h pki
vault write pki/roles/chepia-servers allow_any_name=true
vault write pki/config/urls issuing_certificates="$VAULT_ADDR/v1/pki/ca" \
```

→ TTL de 2 semaine pour tester la rotation de certificats et le bridge CA.

→ Rôle permissif au niveau des noms pour faciliter la mise en place, peut être restreint ultérieurement.

#### 0.3.3.2 Création du CA intermédiaire :

Commandes :

```
vault secrets enable -path=pki_int pki
vault secrets tune -max-lease-ttl=240h pki_int
vault write -format=json pki_int/intermediate/generate/internal \
    common_name="chepia.ch Intermediate Authority" \
    issuer_name="chepia-dot-ch-intermediate" \
    | jq -r '.data.csr' > pki_intermediate.csr
vault write -format=json pki/root/sign-intermediate \
    issuer_ref="root" \
    csr=@pki_intermediate.csr \
    format=pem_bundle ttl="240h" \
    | jq -r '.data.certificate' > intermediate.cert.pem
vault write pki_int/intermediate/set-signed certificate=@intermediate.cert.pem
```

→ TTL 10 jours (doit être plus court que celui de la racine)



### 0.3.3.3 Création du rôle :

- Un rôle définit une politique d'accès au certificats intermédiaire : Le domaine doit être `chepia.ch` ou un sous-domaine comme `www.chepia.ch`, et le temps de vie maximum est de 120 heures.

Commandes :

```
vault write pki_int/roles/chepia-dot-ch \
    issuer_ref="$(vault read -field=default pki_int/config/issuers)" \
    allowed_domains="chepia.ch" \
    allow_subdomains=true \
    max_ttl="120h"
```

### 0.3.3.4 Demander un certificat :

Commande :

```
vault write pki_int/issue/chepia-dot-ch common_name="test.chepia.ch" ttl="24h"
```

→ Crée un certificat valide 24 heures pour le nom de domaine "test.chepia.ch"

## 0.3.4 Configuration de cert-manager

### 0.3.4.1 Activation de cert-manager :

- `microk8s enable cert-manager`

### 0.3.4.2 Configurer l'accès entre Vault et Kubernetes

```
kubectl exec --stdin=true --tty=true vault-0 -- /bin/sh
vault auth enable kubernetes
write auth/kubernetes/config \
    kubernetes_host="https://$KUBERNETES_PORT_443_TCP_ADDR:443"
```

→ La variable globale `$KUBERNETES_PORT_443_TCP_ADDR` à modifier selon votre configuration. L'adresse utilisée par le service principal de Kubernetes peut être consultée avec la commande `microk8s kubectl get services | grep kubernetes`. Dans mon cas, Kubernetes utilise l'adresse 10.152.183.1.

### 0.3.4.3 Configurer l'émetteur de certificats

1. Créer une identité pour communiquer avec l'émetteur. C'est avec cette identité que cert-manager utilisera pour accéder au rôle créé à l'étape précédente :  
`kubectl create serviceaccount issuer`
2. Créer un secret que l'identité `issuer` utilisera pour s'authentifier :

→ On crée pour cela le fichier `issuer-secret.yaml`:

```
apiVersion: v1
kind: Secret
metadata:
  name: issuer-token-lmzpj
  annotations:
    kubernetes.io/service-account.name: issuer
type: kubernetes.io/service-account-token
```

→ Appliquer la config avec la commande `microk8s kubectl apply -f issuer-secret.yaml`.

3. Créer le fichier de configuration de l'émetteur `vault-issuer.yaml`:

```
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: vault-issuer
  namespace: default
spec:
  vault:
    server: http://vault.default:8200
    path: pki_int/sign/chepia-dot-ch
    auth:
      kubernetes:
        mountPath: /v1/auth/kubernetes
        role: issuer
        secretRef:
          name: issuer-token-lmzpj
          key: token
```

→ Appliquer la config avec la commande `microk8s kubectl apply -f vault-issuer.yaml`.

### 0.3.5 Installer Nginx Ingress

- Activer l'addon `ingress` qui permet d'exposer des services gérés par le cluster kubernetes :

`microk8s enable ingress`

- On peut maintenant créer des règles pour exposer des services hors de notre cluster Kubernetes.

### 0.3.6 Mise en place de TLS sur un service

#### 0.3.6.1 Création du service

- Nous allons utiliser un simple serveur web Nginx pour représenter la page d'accueil de notre entreprise. Les étapes suivantes montrent comment créer un stockage pour le code source de notre site et déployer une instance Nginx qui monte ce stockage dans le dossier `/usr/share/nginx/html`.
1. Créer le fichier de config des volumes qui vont stocker nos fichiers HTML (`html_volume.yaml`) :

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-volume-html
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/home/mezmer/Documents/atelier_secu/html"
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pv-claim-html
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: ""
  volumeName: pv-volume-html
resources:
  requests:
    storage: 1Gi
```

2. Créer le fichier de config du serveur (nginx.yaml) :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
        volumeMounts:
        - name: nginx-index-storage
          mountPath: /usr/share/nginx/html
      volumes:
      - name: nginx-index-storage
        persistentVolumeClaim:
          claimName: pv-claim-html
```

3. Appliquer les configs : `microk8s kubectl apply -f html-volumes.yaml`,  
`microk8s kubectl apply -f nginx.yaml`

### 0.3.6.2 Création du certificat du service :

- Création du fichier cert.yaml :

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: chepia-cert
  namespace: default
spec:
  issuerRef:
    name: vault-issuer
  commonName: www.chepia.ch
  secretName: chepia-cert
  dnsNames:
  - www.chepia.ch
```

- Appliquer la config avec la commande `microk8s kubectl apply -f cert.yaml`

### 0.3.6.3 Configuration de l'accès au service

- Créer le fichier `nginx-service.yaml`:

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
spec:
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
  selector:
    app: nginx
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: nginx
spec:
  ingressClassName: nginx
  tls:
    - hosts:
        - www.chepia.ch
      secretName: chepia-cert
  rules:
    - host: www.chepia.ch
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: nginx
                port:
                  number: 80
```

- Appliquer la config avec la commande `microk8s kubectl apply -f nginx-service.yaml`

**0.3.6.4 Test de l'accès au service** Pour pouvoir accéder à notre service de manière sécurisée, il faut le faire par son nom de domaine, car c'est à celui-ci qu'est relié le certificat créé précédemment. Pour cela, on va ajouter la ligne suivante au fichier `/etc/hosts`:

`127.0.0.1 www.chepia.ch`

Ensuite, on peut tester la connexion TLS avec la commande : `curl -kivL 'https://www.chepia.ch'`

```
> curl -kivL 'https://www.chepia.ch'
* Trying 127.0.0.1:443...
* Connected to www.chepia.ch (127.0.0.1) port 443 (#0)
* ALPN, offering h2
* TLSv1.0 (OUT), TLS header, Certificate Status (22):
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.2 (IN), TLS header, Certificate Status (22):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS header, Finished (20):
* TLSv1.2 (IN), TLS header, Supplemental data (23):
* TLSv1.3 (IN), TLS handshake, Encrypted Extensions (8):
* TLSv1.2 (IN), TLS header, Supplemental data (23):
* TLSv1.3 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (IN), TLS header, Supplemental data (23):
* TLSv1.3 (IN), TLS handshake, CERT verify (15):
* TLSv1.2 (IN), TLS header, Supplemental data (23):
* TLSv1.3 (IN), TLS handshake, Finished (20):
* TLSv1.2 (OUT), TLS header, Finished (20):
* TLSv1.3 (OUT), TLS change cipher, Change cipher spec (1):
* TLSv1.2 (OUT), TLS header, Supplemental data (23):
* TLSv1.3 (OUT), TLS handshake, Finished (20):
* SSL connection using TLSv1.3 / TLS_AES_256_GCM_SHA384
* ALPN, server accepted to use h2
* Server certificate:
* subject: CN=www.chepia.ch
* start date: Apr 22 16:51:30 2024 GMT
* expire date: Apr 23 02:52:00 2024 GMT
* issuer: CN=chepia.com Intermediate Authority
* SSL certificate verify result: unable to get local issuer certificate (20), continuing anyway.
* Using HTTP2, server supports multiplexing
* Connection state changed (HTTP/2 confirmed)
* Copying HTTP/2 data in stream buffer to connection buffer after upgrade: len=0
* TLSv1.2 (OUT), TLS header, Supplemental data (23):
* TLSv1.2 (OUT), TLS header, Supplemental data (23):
* TLSv1.2 (OUT), TLS header, Supplemental data (23):
* Using Stream ID: 1 (easy handle 0x55e3c927beb0)
* TLSv1.2 (OUT), TLS header, Supplemental data (23):
> GET / HTTP/2
> Host: www.chepia.ch
> user-agent: curl/7.81.0
> accept: */*
```

Figure 5: Connection à notre service en https avec Curl

On peut voir que le certificat du serveur est bien émit par notre autorité de certification intermédiaire. Par contre la ligne `SSL certificate verify result: unable to get local issuer certificate (20), continuing anyway.` indique que notre CA n'a pas été reconnu. C'est normal, car on a pas encore ajouté le certificat racine de notre autorité à la liste des CA reconnu par notre système d'exploitation. Si on essaye d'accéder à notre service avec un navigateur web, on va rencontrer un avertissement similaire.

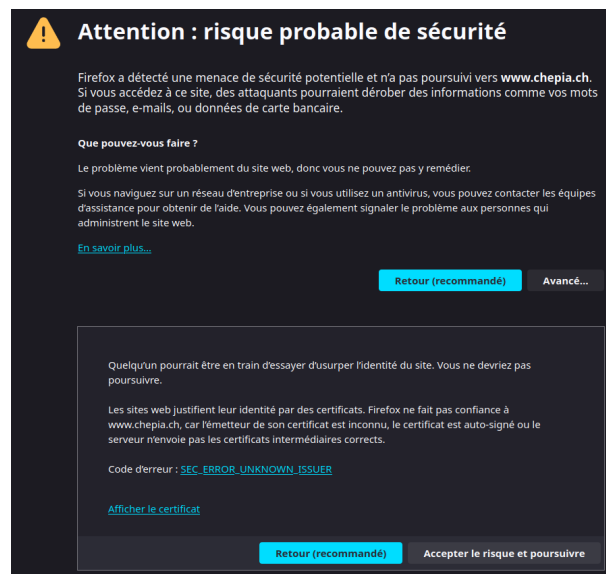


Figure 6: Avertissement de sécurité de Firefox. Notre CA n'est pas reconnu par le navigateur.

Pour remédier à cela, on va chercher le certificat de notre autorité intermédiaire dans l'interface graphique de Vault.

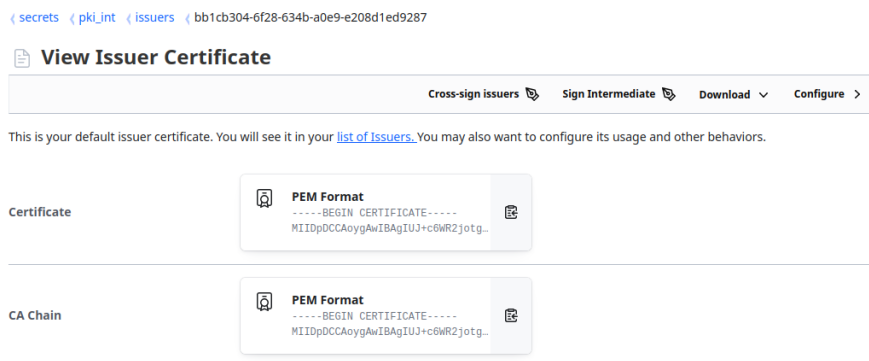


Figure 7: On peut télécharger ou copier le certificat depuis la page secrets -> pki\_int -> issuers -> chepia.ch Intermediate Authority

Ensuite, on crée un sous-répertoire `chepia.ch` dans le répertoire `/usr/local/share/ca-certificates`. On y place le certificat et on met à jour la liste des CAs reconnus avec `sudo update-ca-certificates`.

Il faut aussi ajouter le certificat à la liste des CAs de confiance de votre navigateur. Pour cela, allez dans *préférences*, *Vie privée et sécurité*, *Certificats* et cliquer sur *Afficher les certificats* puis *Importer*. Une fois le certificat importé, on retrouve notre autorité dans la liste des CAs de confiance.



Figure 8: Liste des CAs de confiance dans Firefox

Si on retourne sur `www.chepia.ch`, l'alerte a disparu.

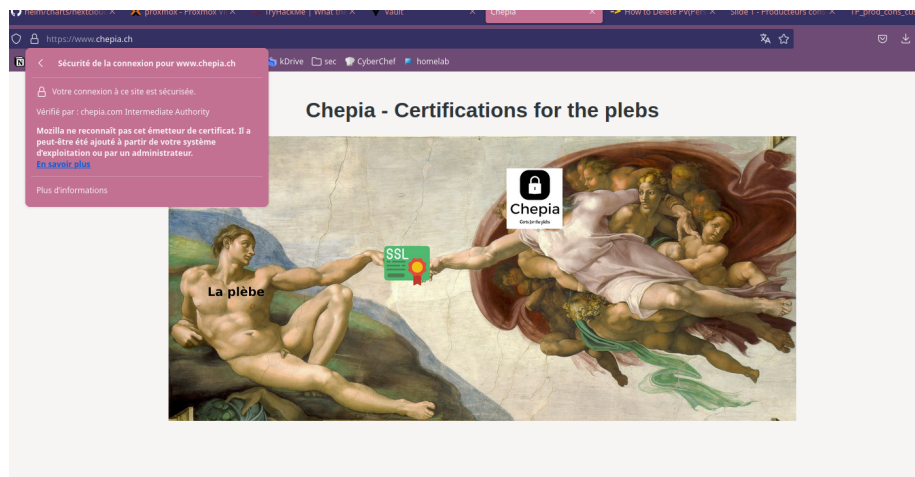


Figure 9: Le point d'exclamation a disparu du cadenas, on peut utiliser le service de manière sécurisée.



### 0.3.7 Mise à jour des certificats

#### 0.3.7.1 Suppression des certificats expirés

```
vault write pki_int/tidy tidy_cert_store=true tidy_revoked_certs=true
```

#### 0.3.7.2 Rotation du certificat racine

```
vault write pki/root/rotate/internal \
    common_name="chepia.com" \
    issuer_name="root-2"
```

```
vault write pki/roles/chepia-servers allow_any_name=true
```

**0.3.7.3 Création d'un pont entre le nouveau certificat racine et l'ancien** Cette étape permet aux services qui n'ont pas communiqué avec le CA depuis longtemps et doivent mettre à jour leur certificat racine. Le pont permet à ses service de faire confiance à notre nouveau certificat racine.

```
cd vault
```

```
vault write -format=json pki_int/intermediate/cross-sign \
    common_name="chepia.com" \
    key_ref="$(vault read pki_int/issuer/root-2 \
| grep -i key_id | awk '{print $2}')" \
| ./jq -r '.data.csr' \
| tee cross-signed-intermediate.csr

vault write -format=json pki_int/issuer/root/sign-intermediate \
    common_name="chepia.com" \
    csr=@cross-signed-intermediate.csr \
| ./jq -r '.data.certificate' | tee cross-signed-intermediate.crt

vault write pki/intermediate/set-signed \
    certificate=@cross-signed-intermediate.crt
```

#### 0.3.7.4 Remplacer l'émetteur par défaut

```
vault write pki/root/replace default=root-2
```

### 0.3.7.5 Signer l'autorité intermédiaire avec notre nouveau certificat racine

```
cd vault
```

```
vault write -format=json pki_int/intermediate/cross-sign \
  common_name="chepia.com Intermediate Authority" \
  key_ref="$(vault read pki_int/issuer/$(vault read -field=default pki_int/config/issuers
| grep -i key_id | awk '{print $2}')" \
  | ./jq -r '.data.csr' \
  | tee cross-signed-intermediate.csr
```

```
vault write -format=json pki/issuer/root-2/sign-intermediate \
  common_name="chepia.com Intermediate Authority" \
  csr=@cross-signed-intermediate.csr \
  | ./jq -r '.data.certificate' | tee cross-signed-intermediate.crt
```

```
vault write pki_int/intermediate/set-signed certificate=@cross-signed-intermediate.crt
```

## 0.3.8 Mise en place du service ACME

### 0.3.8.1 terminal :

```
vault write /sys/mounts/pki_int/tune \
  passthrough_request_headers="If-Modified-Since" \
  allowed_response_headers="Last-Modified,Location,Replay-Nonce,Link"
```

### 0.3.8.2 GUI :

- Aller à la page de configuration de l'autorité intermédiaire.
- Ajouter l'URL `http://127.0.0.1:8200/v1/pki_int` au champ AIA path et Mount's API path
- Cocher "Enable ACME"
- Sauver la configuration

### 0.3.9 Connecter Cert-manager au service ACME

#### 0.3.9.1 Ajout d'un émetteur

- Création du fichier de configuration acme.yaml

```
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: chepia-ch
spec:
  acme:
    server: http://127.0.0.1:8200/v1/pki_int/acme/directory
    caBundle: <intermediate CA cert in PEM format encoded in base64>
    privateKeySecretRef:
      name: issuer-token-lmzpj
    solvers:
      - selector: {}
        http01:
          ingress:
            ingressClassName: nginx
```

- Activation de la config avec `microk8s kubectl apply -f acme.yaml`

## 0.4 Glossaire

- **Kubernetes** : Système d'orchestration de conteneurs qui permet de créer facilement une infrastructure virtuelle en automatisant le déploiement d'applications, la configuration du réseau et des services, etc.
- **Certificat** : Dans le contexte de la sécurité informatique, un certificat est un document numérique permettant d'identifier et d'authentifier un utilisateur ou un service. Il contient aussi les informations nécessaires à l'établissement d'une communication sécurisée (clef publique, protocoles de chiffrements utilisés, etc).
- **Infrastructure à clef publique ou PKI (Public Key Infrastructure)** : Une PKI est un ensemble de processus basés sur l'utilisation de certificats qui permettent de chiffrer et d'authentifier les communications entre utilisateurs et services ou de service à service.
- **Politique de certification** : Ensemble de décisions qui définissent les caractéristique des certificats émis. La politique de certification définit le temps de vie des certificats, la chaîne de confiance, les politiques de révocations, etc.
- **Autorité de certification ou CA (Certificate Authority)** : Une autorité de certification est un organisme qui agit comme tiers de confiance. La CA est responsable de l'émission de certificats et de leur validation. Pour être considérée comme un tiers de confiance valide, l'autorité doit être reconnue par les principaux navigateurs et systèmes d'exploitation.
- **Chaîne de confiance** : Hiérarchie utilisée par les certificats pour vérifier la validité de l'émetteur.

## 0.5 Références

- <https://developer.hashicorp.com/vault/tutorials/secrets-management/pki-engine?variants=vault-deploy%3Aselfhosted>
- <https://developer.hashicorp.com/vault/tutorials/kubernetes/kubernetes-cert-manager>
- <https://developer.hashicorp.com/vault/docs/concepts/seal>
- [https://fr.wikipedia.org/wiki/Autorit%C3%A9\\_de\\_certification](https://fr.wikipedia.org/wiki/Autorit%C3%A9_de_certification)
- [https://fr.wikipedia.org/wiki/Certificat\\_%C3%A9lectronique](https://fr.wikipedia.org/wiki/Certificat_%C3%A9lectronique)
- [https://en.wikipedia.org/wiki/Certificate\\_policy](https://en.wikipedia.org/wiki/Certificate_policy)
- [https://en.wikipedia.org/wiki/Key\\_management](https://en.wikipedia.org/wiki/Key_management)
- <https://www.digicert.com/fr/what-is-pki>
- <https://www.ssldragon.com/blog/best-practices-to-store-the-private-key/>
- <https://www.thesslstore.com/blog/what-is-a-key-management-service-key-management-services-explained/>
- [https://cheatsheetseries.owasp.org/cheatsheets/Key\\_Management\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Key_Management_Cheat_Sheet.html)