RAPPORT PROJET CRYPTOGRAPHIE

Signature de certificat signant des mails

Résumé

Création d'une ACR et d'une ACI afin de signer les demandes de certificat et ainsi pouvoir signer des mails.

Loïc SORROCHE
Mathias COUSSEAU

Table des matières

Le sujet :

Résumé :

Ce projet consiste à créer une autorité de certification (AC) et une autorité d'enregistrement (AE) permettant de générer et délivrés des certificats pour des utilisateurs désirant signer leurs emails.

Objectifs :

- Le processus consiste à créer une autorité de certification racine (ACR) avec son certificat auto-signé. Ensuite, une autorité de certification intermédiaire (ACI) est créée avec son certificat émis par l'ACR, qui émettra les certificats des utilisateurs.
- Une page web de l'AE est créée pour permettre aux utilisateurs de soumettre une demande de signature de certificat (CSR) avec leur adresse email et une page de révocation sera mise en place pour permettre aux utilisateurs de révoquer leur certificat.
- Les utilisateurs génèrent une paire de clés et la CSR correspondante.
- L'AE vérifie l'email et la CSR de l'utilisateur, demande la génération du certificat à l'ACI et le met à disposition de l'utilisateur.
- Ensuite, des tests sont effectués pour vérifier si les certificats non valides sont détectés.
- Puis monter une attaque en créant une fausse AC pour tester si celle-ci est détecté.
- Enfin, un serveur OCSP est configuré pour indiquer l'adresse du serveur et tester une révocation.

Choix des technologies

- Plateforme Web : Le front-end a été fait en html et css, choix basique. Le back-end a lui été codé en python pour sa simplicité et sa rapidité de mise en place.
- Génération des certificats : Nous utilisons la CLI bash « openssl » pour générer nos certificats, choix imposé mais quasi-obligatoire pour ce genre de tâches. Plus précisément nous utilisons des fichiers de configuration openssl avec la commande « openssl ca », nous détaillerons plus tard. Nous choisissons les durées suivantes pour nos certificats : RCA = 10ans, ICA = 3 ans, User = 90jours. Ainsi nous sécurisons un peu plus nos certificats en leur accordant une durée de vie limité, tout en respectant la "pyramide" entre la RCA et les certificats finaux.
- Choix du type de clés : Nous utilisons des clés basées sur des courbes elliptiques pour générer notre ACR et ACI pour leur taille réduite et leur niveau de sécurité reconnue.
- Algorithme de signature des certificats : Nos certificats signent avec les mails avec ecdsa-SHA512, un choix sécurisé et compatible avec tous les client mail récents.
- Client Mail de test : Nous avons effectué nos tests avec le client mail « Thunderbird », simple d'utilisons et installé par défaut sur Ubuntu Linux.

Suivi du sujet :

1) Pour créer l'ACR on doit :

- Créer une clé privée de préférence elliptique, car elle permet des clefs plus puissantes car basé sur des courbes elliptiques basée sur des algorithmes plus petits pour générer des clés qui sont exponentiellement plus fortes, de plus avec un algorithme plus petit on a une augmentation des performances

Commandes :

```
openssl ecparam -genkey -name prime256v1 -out private.key
```

- A présent on chiffre et salt la clé :

```
openssl aes-256-cbc -in private.key -salt -out private.enc
```

- on génère le .csr à partir de notre précédente clé :

```
openssl req -new -key private.enc -aes256 -out rootCA.csr #genere le csr
```

- avec notre .csr on créer notre .crt :

```
Openssl ca -selfsign -keyfile private.key -config caVF.cnf -in rootCA.csr -out rootCA.pem -days 3650 -extensions v3_ca
```

On possède à présent notre RCA, nous choisissons 10ans comme durée de vie par simplicité. Nous mettons de côté notre clé privé et gardons le certificat.

2) Création de l'ICA :

- On créer un clé privé elliptique :

```
openssl ecparam -name prime256v1 -genkey -noout -out ica.key
```

- On créer le .csr associer :

```
openssl req -new -key ica.key  -out ica.csr
```

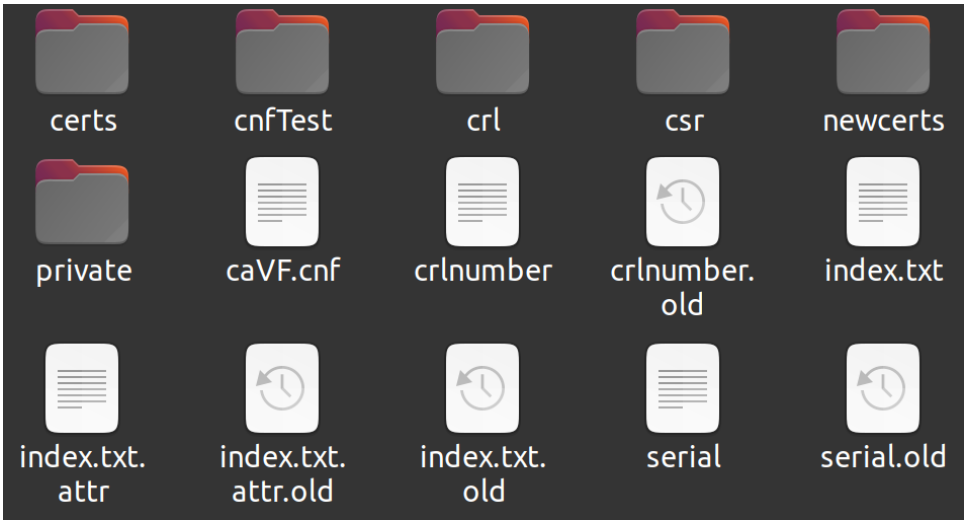
- Puis à partir du .csr et du fichier de configuration on créer le .cert :

```
openssl ca -batch -config caVF.cnf -CA rootCA.pem -Cakey private.enc -extensions v3_intermediate_ca -days -in ica.csr -out ica.cert
```

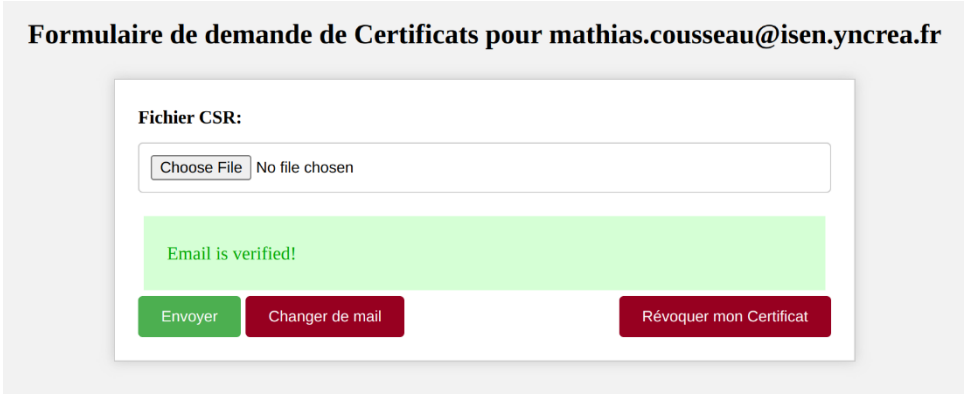
Le fichier de configuration (.cnf) donne les règles de configuration de notre certificat (voir dans myrootca/CA/ca.cnf)

Ce fichier nous permet d'automatiser la création de certificat. Nous y configurons l'emplacement des dossiers et fichiers nécessaires, comme l'emplacement des nouveaux certificats, ou le certificat et la clé de l'ICA afin qu'il puisse signer les nouveaux certificats. Nous y ajoutons des extensions (comme « v3_ca » pour la RCA, « v3_intermediate_ca » pour l'ICA ou « smime » pour les certificats utilisateurs) afin de pouvoir personnaliser au besoin les certificats générés.

Un fois notre ICA fait, on se retrouve avec l'arborescence suivante :



3) Nous codons rapidement une interface web avec la librairie Flask pour que les utilisateurs puissent soumettre leur CSR ainsi que leur mail afin de faire signer leur nouveau certificat par notre ACR.



(Screenshot effectué à la fin du projet)

4) Coté utilisateur on générer une paire de clés :

- création d'une clé privée

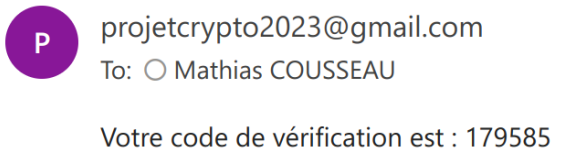
```
openssl genpkey -algorithm RSA -out private.key
```

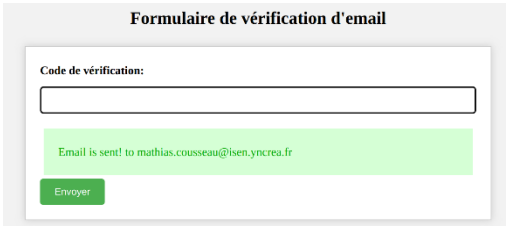
- création du .crs à partir de la clé précédente

```
openssl req -new -key private.key -out csr.csr
```

5) Nous codons alors 2 systèmes :

- Un premier système qui envoie un mail à l'adresse renseignée par l'utilisateur afin de vérifier qu'il s'agit bien de lui.





- Un deuxième vérifiant le champ « email » de la CSR afin de le comparer à celle de l'utilisateur connecté.

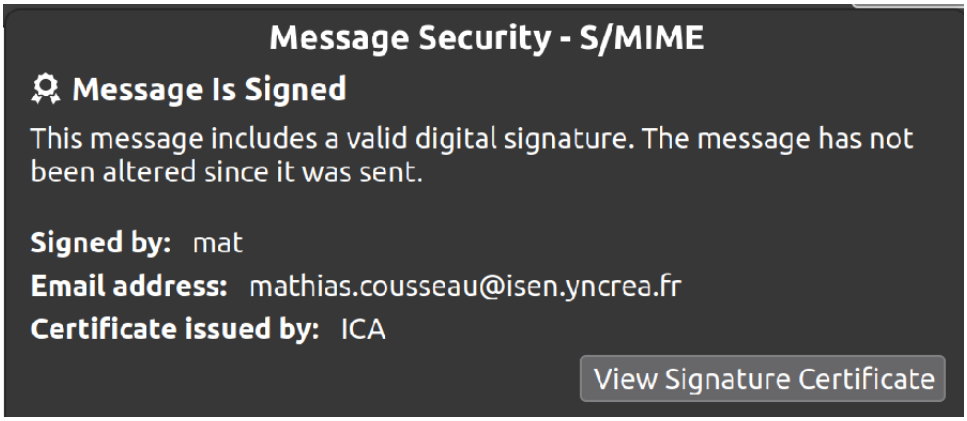
```
CSRemail=$(openssl req -in $1 -noout -subject -nameopt multiline
| awk -F= '/email/ {gsub(/^[ \t]+/,"",$2); print $2}')
if [ "$CSRemail" == "$2" ]; then
```

En cas de succès de ces 2 systèmes, la génération du certificat, signé par l'ACI, est fait et ce dernier est renvoyé à l'adresse mail de l'utilisateur afin qu'il puisse la télécharger. En plus de ce certificat, nous mettons a disposition toute la chaine de certification pour que le client web puisse faire le lien entre la RCA et le certificat utilisateur. À savoir qu'un utilisateur ne peut avoir qu'un certificat actif pour son adresse mail à la fois.

```
def sendCert(email):
    cert=findCert(email)
    print("cert=",cert)
    if cert == "ERROR":
        return "ERROR finding cert in server"
    else:
        sendEmail(email,cert)
        return "Certificat envoyé à l'adresse mail : " + email
```

Exemple de certificat envoyé :

6) Nous uploadons notre RCA dans les authorités de confiance de ThunderBird, puis notre certificats utilisateur dans nos certificats perso. Nous envoyons un mail signé et obtenons ceci :



Notre certificat fonctionne donc correctement pour la signature.

7) Après avoir de générer des certificats non valides et de signer avec, nous obtenons constamment la même erreur:

Nous en concluons donc que ces attaques sont détectées par notre client mail et nous empêche de signer un mail avec un certificat n'étant pas issu d'une de nos autorités de confiance ou lorsqu'il est expiré.

8) Nous essayons cette fois de générer une CA à partir d'un certificat utilisateurs et ainsi pouvoir de générer un nouvelle certificat utilisateur à partir d'une fausse CA.

```
 1227  openssl x509 -req -in ./fakeCA.csr    -CA ../crypto.pem   -C
Akey ../key.pem -CAcreateserial -out fakeCA.pem -days 365 -extfile
../../CA/caVF.cnf  -extensions v3_intermediate_ca
```

Création de la CA

```
 1228  openssl x509 -req -in ./fakeCert.csr  -CA fakeCA.pem   -CAke
y key.pem -CAcreateserial -out ./certs/fakeCert.pem -days 365 -extf
ile ../../CA/caVF.cnf  -extensions smime
```

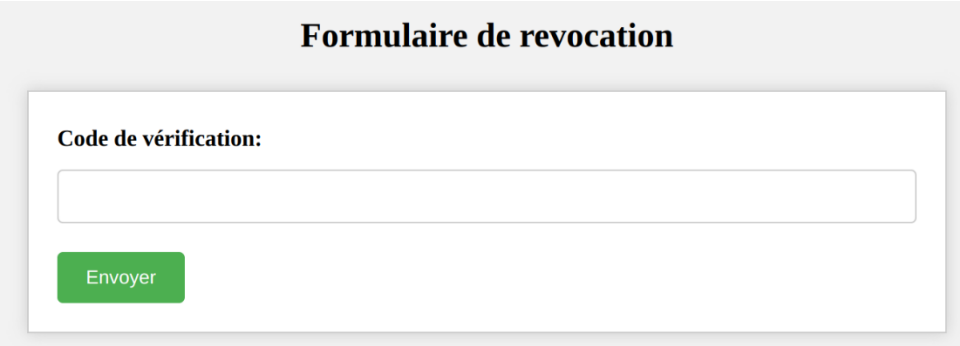
Création du cert

Après quelques recherches nous convenons qu'il faut rajouter la directive de configuration "pathlen: 0" sur notre extension ICA puis la regénérée. Cela indique que notre ICA ne peut pas générer de CA supplémentaire.

Ainsi lorsqu'on donne à Thunderbird notre faux certificat utilisateur, créé par notre fausse CA, ainsi que toute la chaine de certification jusqu'à notre RCA d'origine, notre client ne reconstruit pas le chemin et agit comme s'il s'agissait d'un certificat sans CA connue. Nous obtenons donc la même erreur que précédemment et déduisons que l'attaque est un échec.

9) Nous codons une page permettant de saisir un code à usage unique pour révoquer un certificat déjà créer ayant pour sujet l'adresse mail connectée. Ce code est transmis lors de la génération du certificat dans le mail à l'utilisateur. Une fois utilisé le certificat est noté comme révoqué, supprimé

de la base du serveur et le code n'est plus utilisable. L'utilisateur peut alors régénérer un nouveau certificat.



10) Pour faire le serveur ocsp nous utilisons la commande "openssl ocsp" en indiquant le fichier "index.txt" où sont notés tous les certificats générés par notre fichier de configuration, ainsi que ceux révoqués. Nous créons un petit script, facilitant son lancement :

```
openssl ocsp -index ../CA/index.txt -port 8888 -rsigner ocsp.crt -rkey key.pem -CA ocsp-chain.pem -text
```

Cette commande nécessite cependant de créer un certificat pour le sujet ocsp, ce que nous faisons donc à la manière de tous nos précédents certificats, en créant une nouvelle extension dans le fichier configuration. Nous profitons de cette occasion pour rajouter dans l'extensions "smime" l'addresse du serveur OCSP pour y faire les requêtes.

```
authorityInfoAccess = OCSP;URI:http://localhost:8888
```

Lors de nos tests, le serveur répond correctement aux requêtes manuelles (à conditions qu'il se soit mis à jour en relisant le fichier index.txt), et nous pouvons également observer les requêtes faites avec notre fichier de log du serveur.

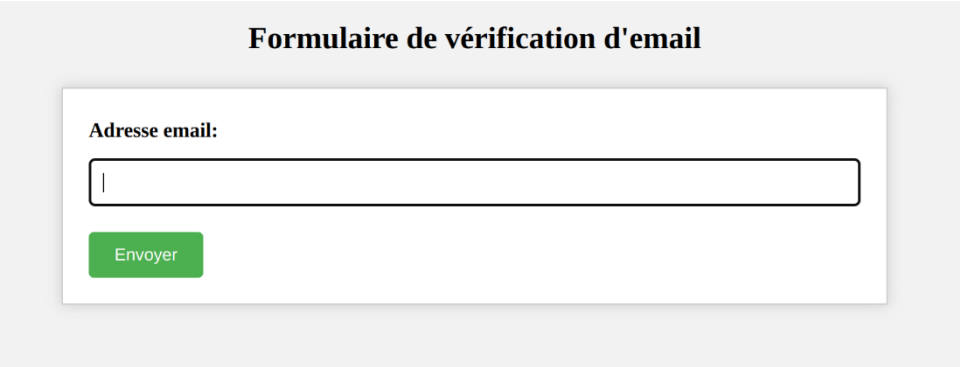
```
❯ openssl ocsp -CAfile ../CA/private/new/ica.pem -issuer ../CA/priv
ate/new/ica.pem -cert crypto.pem -url http://localhost:8888 -resp_t
ext -noverify
OCSP Response Data:
    OCSP Response Status: successful (0x0)
    Response Type: Basic OCSP Response
```

```
crypto.pem: good
        This Update: May  1 22:56:49 2023 GMT
```

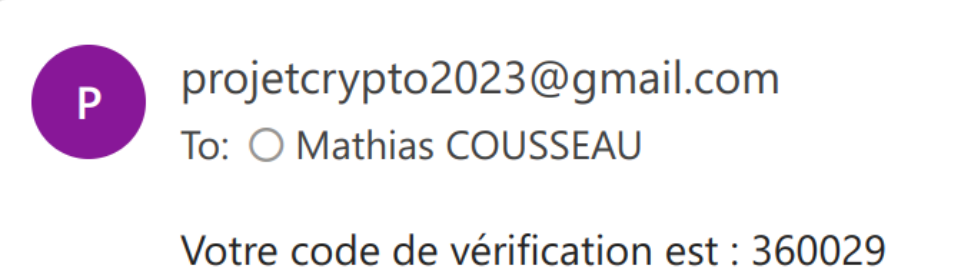
Cependant nous pensons que notre client mail possède un cache pour nos certificats. En effet, ThunderBird effectue une requête au serveur OCSP lors de la première utilisation du certificat et puis plus rien (du moins dans les 2 heures suivantes). Nous considérons donc que notre serveur fonctionne mais soulignons qu'il faut attendre un moment avant que les réponses soient mises à jour.

Déroulement d'une demande de certificats à la suite d'une soumission de CSR:

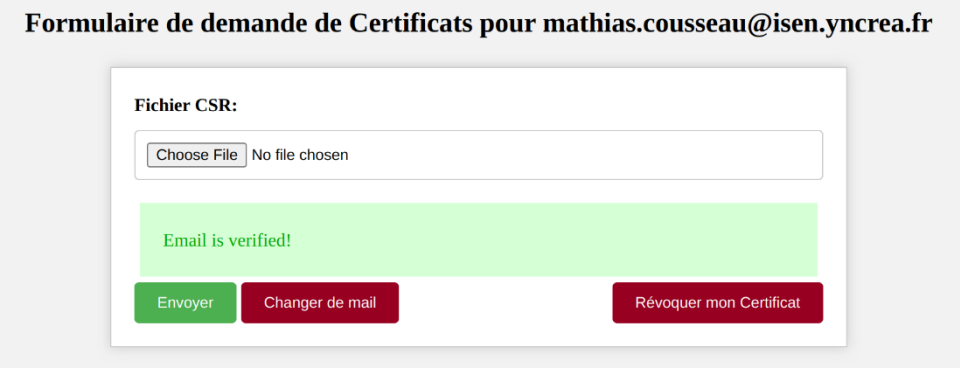
1) L'utilisateur génère sa paire de clé ainsi que sa csr, en précisant bien l'adresse mail
2) L'utilisateur se rend sur le site



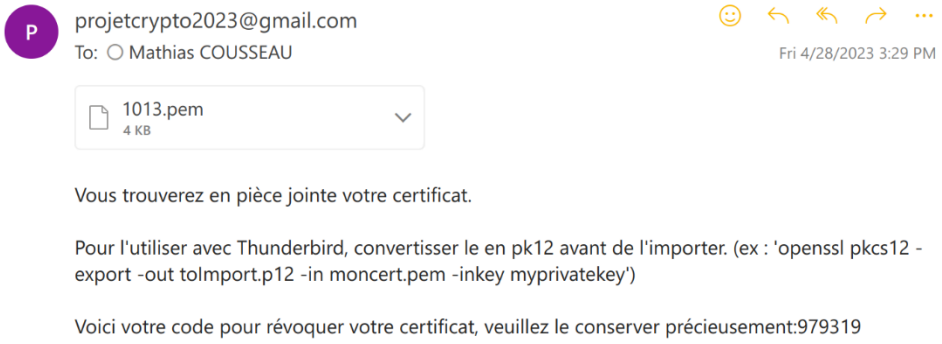
3) Il rentre son adresse mail et reçoit alors un mail avec un code afin de vérifier qu'il possède effectivement cette adresse mail



4) Une fois le code renseigné, l'utilisateur pourra accéder à la page lui permettant de soumettre sa csr.



5) Après avoir soumis sa demande csr, l'utilisateur recevra par mail, s'il n'y a pas d'erreur lors de la génération, son nouveau certificat avec les instructions pour l'utiliser dans Thunderbird et le code de révocation.



6) Pour révoquer son certificat, l'utilisateur devra cliquer sur Révoquer mon certificat après s'être connecté, il devra alors renseigner son code obtenu dans son mail. Si le code est bon, le certificat sera alors noté comme révoqué et supprimer du serveur. L'utilisateur pourra alors le regénérer en suivant la démarche précédemment expliqué.

À noter que le site dispose d'un système de code d'erreur permettant à l'utilisateur de connaitre l'origine de l'erreur. Donc en cas d'erreur l'utilisateur sera capable de corriger la partie défaillante.

Conclusion:

Difficultés rencontrées :

Tout d'abord, nous avons mis du temps à comprendre comment fonctionnait la commande "openssl ca" ainsi que les particularités de rédaction du fichier de configuration openssl. La CLI openssl a elle aussi été capricieuse, avec souvent des erreurs liées à l'architecture de notre projet, chose nouvelle pour nous.

Nous avons eu aussi à réfléchir à comment stocker l'OTP et autres informations entre les pages, nous avons finalement opté pour un token jwt, preuve d'intégrité, chiffré à partir d'une clé et d'un algorithme maison pour garantir la confidentialité, notamment de l'OTP.

Comme expliqué auparavant, nous avons un souci avec le serveur OCSP et ThunderBird, nous soupçonnons que ce dernier possède un cache interne mais n'avons pas pu plus faire de recherche sur le sujet.

Points d'améliorations :

- Une amélioration de la gestion d'erreur pour encore plus de précision
- Des vérifications plus précises et spécifiques sur les champs de la csr (exe mple : le nom de l'entreprise doit matcher avec celui de la RCA, etc…)
- Imposer un niveau de sécurité de l'algorithme de la clé de sécurité de l'utilisateur
- Ajout d'une justification de l'utilisateur lors de la révocation de certificat.
- Vérification des inputs sur la plateforme Web (Path traversal, RCE, etc..)
- Véritable random pour notre OTP, actuellement pseudo-random uniquement
- Si déploiement, revoir les droits de chaque utilisateur pour qu'on ne puisse pas accéder à n'importe quel fichier sur le serveur.
- Faire fonctionner correctement le serveur OCSP avec le client mail Thunderbird.

- Améliorer le token jwt, stocker les mots de passes et clé du système dans des fichiers tiers.

Conclusion :

Ce projet nous a permis de découvrir fonctionnement d'une autorité de certification, la façon dont on crée et délivre des certificats, et de mieux comprendre le fonctionnement des signatures de mail, tout en passant par des réflexions sur attaque possible sur cette surface. Grace à ce projet nous avons appris à utiliser des outils comme Thunderbird, Flask ou encore de prendre en main la CLI openssl, indispensable pour ce type de projets.

Biblio :

https://www.entrust.com/fr/resources/certificate-solutions/learn/elliptic-curve-cryptography-ssl

https://chat.openai.com/

https://kctheservant.medium.com/two-typical-setups-of-fabric-ca-server-using-a-self-generated-root-ca-or-a-given-intermediate-ca-41b47b2e5614