

Rapport de cryptographie

Nos choix techniques

1. Environnement de travail & langages

Pour réaliser le projet de cryptographie, nous avons décidé d'utiliser l'IDE *PyCharm*. Cet IDE est fait pour le langage de programmation Python, qui est par ailleurs le langage que nous avons utilisé pour créer notre entité de certification.

Pour nous, Python était une évidence pour sa simplicité qui nous a fait gagner un temps précieux. De plus, ce langage est bien documenté ce qui nous a été utile lors de nos recherches pour les différentes bibliothèques à utiliser.

Pour nos pages web, nous avons utilisé du *HTML* et du *CSS*.

Au niveau du travail d'équipe, *GitHub* a été le choix le plus évident par ses fonctionnalités et nos habitudes de travail.

Afin de tester nos certificats de mails pour notre projet, nous avons utilisé le logiciel *Thunderbird*.

2. Bibliothèques utilisées

Voici la listes des bibliothèques utilisées pour le projet :

- **Flask**, nous a permis de créer notre serveur. Cette bibliothèque permet de faire de la redirection facilement. C'est une référence pour faire du serveur web en *Python*.
- **Random**, permet de générer un nombre aléatoire. Nous avons utilisé cela afin de générer notre code de vérification qui est envoyé par mail à notre l'utilisateur. Notre code est composé de 6 chiffres.
- **Smtplib**, rend possible la connexion à des serveurs *SMTP* sous *Python*.
- **Ssl**, rend possible l'utilisation de *SSL* avec *Python*. Dans le cadre de notre projet, l'utilisation du *SSL* dans la communication avec le serveur est important pour assurer la sécurité de celle-ci.
- **Subprocess**, rend possible l'exécution de code dans un Shell avec *Python*. Cela nous a été très utile pour rendre possible l'exécution de commande comme « *ssl req* » depuis notre serveur.
- **Zipfile**, pour rendre le téléchargement des fichiers « *.crt* » et « *.key* » possible, nous avons été contraints d'utiliser cette bibliothèque qui permet de créer un fichier « *.zip* » en local pour rendre le téléchargement plus aisé par la suite.
- **Multiprocessing**, permet de lancer plusieurs process en même temps.
- **Time**, nous permet de mieux gérer le processing dans notre code avec des sleeps.

Nos choix de sécurités

1. ACR

Tout d'abord on crée une paire de clé pour notre ACR avec la commande openssl suivante :

```
openssl ecparam -genkey -name prime256v1 -out ACR/root_ca.key
```

Notre paire de clé aura les caractéristiques suivantes :

- Elle est créée à partir d'une courbe elliptique *NIST P-256* en raison de sa sécurité, de sa rapidité et de sa taille de clé plus petite.
- Elle est enregistrée dans le répertoire *ACR* sous le nom de *root_ca.key*

Puis on crée le CRT de notre ACR avec la commande openssl req suivante :

```
openssl req -new -x509 -key ACR/root_ca.key -out  
ACR/root_ca.crt -days 365
```

Notre certificat aura alors les caractéristiques suivantes :

- Une durée de validité de 1 an
- Auto-signé avec une clé privée située dans l'*ACR* sous le nom de *root_ca.key*
- Notre certificat est enregistré dans le répertoire de l'*ACR* sous le nom de *root_ca.crt*

Notre certificat va alors nous permettre de signer d'autres certificats et ainsi créer une chaîne de confiance.

2. ACI

Tout d'abord on crée une paire de clé pour notre ACI avec la commande openssl suivante :

```
openssl ecparam -genkey -name prime256v1 -out  
ACI/intermediate_ca.key
```

Notre paire de clé aura les caractéristiques suivantes :

- Elle est créée à partir d'une courbe elliptique *NIST P-256* en raison de sa sécurité, de sa rapidité et de sa taille de clé plus petite.
- Elle est enregistrée dans le répertoire *ACI* sous le nom de *intermediate_ca.key*

Puis on crée le CSR de notre ACI avec la commande openssl req suivante :

```
openssl req -new -key ACI/intermediate_ca.key -out  
ACI/intermediate_ca.csr
```

Maintenant, on crée le CRT de notre ACI avec la commande openssl req suivante :

```
openssl x509 -req -in ACI/intermediate_ca.csr -CA  
ACR/root_ca.crt -CAkey ACR/root_ca.key -CAcreateserial -out
```

```
ACI/intermediate_ca.crt -days 730 -extfile  
ACI/intermediate_ca.cnf -extensions v3_intermediate_ca
```

Notre CRT aura les caractéristiques suivantes :

- Signé avec le certificat et la clé privée de l'ACI
- Une durée de validité de 2 ans

3. Certificats

Notre utilisateur peut créer ses propres certificats mais cela doit se faire en toute sécurité et sans la possibilité d'en créer de faux. Pour cela, nous avons mis en place plusieurs barrières de sécurité.

Tout d'abord, après avoir rempli les informations pour la création de son certificat, on va vérifier l'identité de la personne en envoyant par mail un code généré aléatoirement à l'adresse mail donnée par l'utilisateur. Ce code est à 6 chiffres.

L'utilisateur devra alors entrer ce code dans un champ situé sur la page *vérification.html*. Ce formulaire est protégé des injections et ne peut pas être bypassé ce qui rend impossible la création de certificat non consentie.

Une fois cette étape de vérification effectuée, le serveur va alors créer un *CSR*. Pour se faire, on génère notre clé privée en utilisant l'algorithme *RSA* avec de l'*AES 256* comme algorithme de chiffrement pour protéger notre clé privée. On se contente de cet algorithme de chiffrement car il est rapide et simple d'utilisation. De plus, on part du principe que cette clé ne sera utilisée que pour un an. En plus de ça, on protège notre clé privée par un mot de passe gardé secret.

Nos résultats

1. Création de l'ACR et du certificat auto-signé

Comme décrit précédemment, la création de notre ACR et de notre certificat auto-signé n'a pas posé de problèmes et est fonctionnelle. Vous trouverez le résultat en annexe.

2. Création de notre ACI

Comme décrit précédemment, la création de notre ACI n'a pas posé de problèmes et est fonctionnelle.

3. Création de la page web de notre AC

Pour notre page web nous sommes restés simple avec l'utilisation de page HTML et d'un code CSS. Il y a au total 8 pages html pour rendre l'expérience utilisateur plus agréable.

Voici la liste de nos pages :

- *error.html* : permet d'indiquer à l'utilisateur une erreur lors de la saisie du code de vérification.
- *form.html* : est la page formulaire qui permet à l'utilisateur de saisir ses données pour la création du certificat.
- *home.html* : cette page est notre page de home où l'utilisateur peut choisir entre créer un certificat ou en révoquer un.

- `revoke.html` : permet à l'utilisateur de révoquer un certificat existant.
- `revoke_error.html` : indique à l'utilisateur une erreur dans la saisie des informations pour la révocation du certificat.
- `revoke_success.html` : indique à l'utilisateur que le certificat a bien été révoqué.
- `success.html` : indique à l'utilisateur que le certificat a bien été créé.
- `verification.html` : cette page permet à notre AC de vérifier qu'il s'agit bien d'une demande de certificat effectuée par l'utilisateur concerné en demandant à l'utilisateur de saisir un code de vérification reçu par mail.

4. Création de la paire de clé utilisateur et de la CSR

En théorie, la paire de clé est créée coté client, mais pour des raisons pratiques, la paire est générée dans le code Python du serveur avec les informations fournies par l'utilisateur. Elle sera alors disponible en téléchargement lorsque le certificat sera généré.

La commande utilisée pour générer notre clé privée est la suivante :

```
openssl genpkey -algorithm RSA -out $cn.key -aes256 -pass  
pass:"isen"
```

Comme décrit précédemment dans la section sécurité on utilise du *RSA* et de l'*AES 256*.

On génère la *CSR* avec la commande suivante :

```
openssl req -new -key $cn.key -out $cn.csr -subj  
"/emailAddress=$email/CN=$cn/O=$org/OU=$unit/C=$country/ST=$state/L=$city" -passin pass:"isen"
```

On pourra remarquer qu'on utilise la clé privée de l'utilisateur pour signer notre *CSR*. De plus, on remplit notre *CSR* avec toutes les informations saisies par l'utilisateur.

Dans notre script, tout cela se passe avec cette commande :

```
# Création du CSR  
cmd = f"./static/createCSR.sh '{name}' '{email}' '{country}'  
'{state}' '{city}' '{org}' '{unit}' '{cn}'"  
subprocess.check_output(cmd, shell=True)  
print("CSR créé")
```

Les deux commandes SSH sont stockées dans le fichier *createCSR.sh* qui permet de rendre notre script plus lisible.

5. Vérification de l'adresse mail utilisateur et de la CSR

On vérifie l'adresse mail utilisateur grâce à un code généré aléatoirement puis envoyé sur son adresse mail. On utilise la fonction suivante afin de créer un code unique à chaque fois que cela est nécessaire:

```
def generate_validation_code():  
    code = ""  
    for i in range(6):  
        code += str(random.randint(0, 9))  
    return code
```

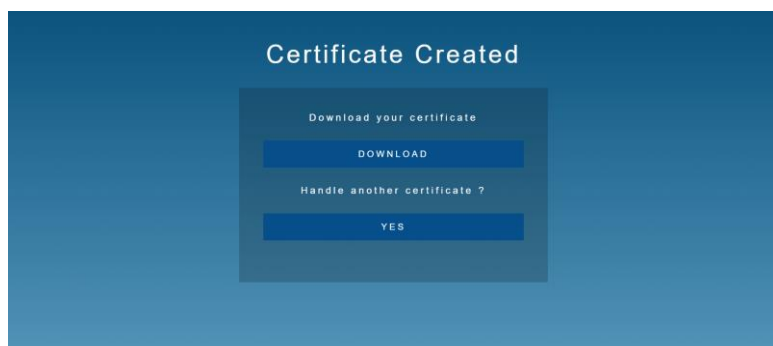
Thomas JAXEL
Hugo SAVY

Vous trouverez en annexe le mail avec le code de vérification que l'utilisateur reçoit.

Une fois ce code récupéré par l'utilisateur il peut le saisir dans le champ prévu à cet effet dans la page *verification.html*. Si le code est bon l'utilisateur est bien redirigé vers la page *success.html* et sa *CSR* est alors créée.

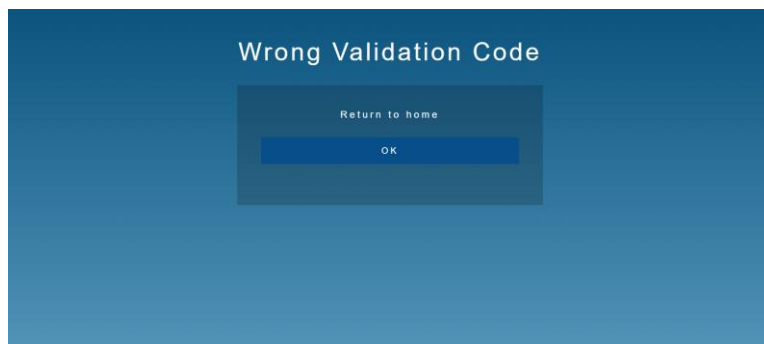


Verification.html



success.html

Si le code saisi par l'utilisateur n'est pas correct, on le dirige vers la page *error.html*.



error.html

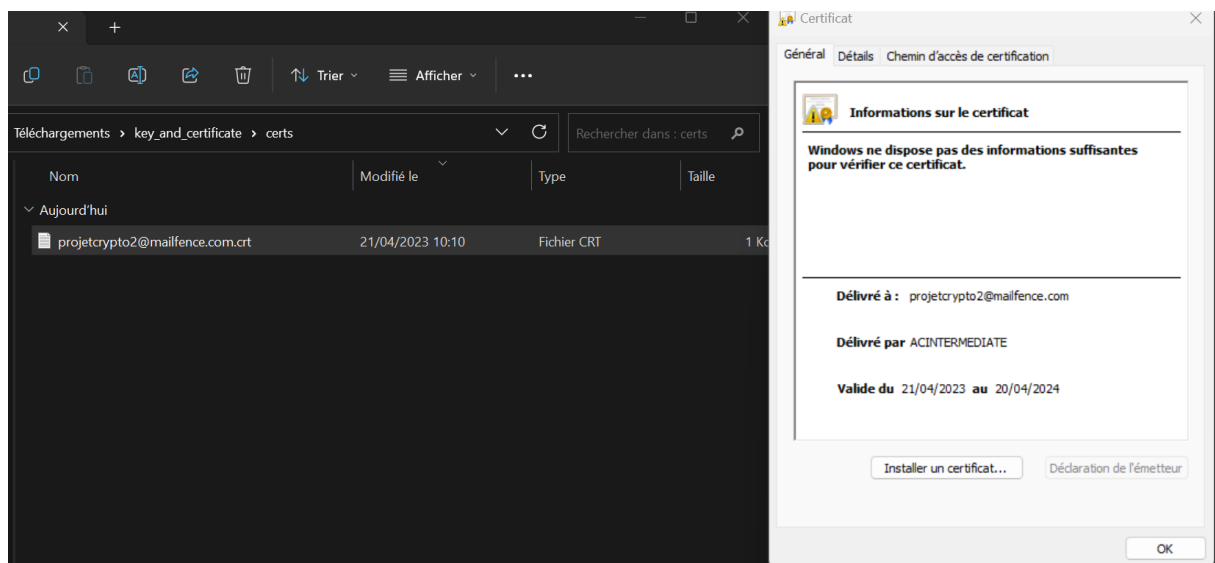
Pour vérifier la *CSR* de l'utilisateur, on compare les informations saisies par notre utilisateur avec les informations contenues dans notre *CSR*. Vous trouverez le morceau de script correspondant en annexe.

6. Génération du certificat et téléchargement des certificats disponibles pour l'utilisateur

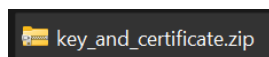
Pour créer le certificat de l'utilisateur, on a juste à prendre notre *CRT* et à exécuter la commande ci-dessous.

```
# Création du CRT
crt_file = "certs/" + cn + ".crt"
cmd = "openssl x509 -req -in " + csr_file + " -CA
ACI/intermediate_ca.crt -CAkey ACI/intermediate_ca.key -
CAcreateserial -out " + crt_file + " -days 365 -sha256 -passin
pass:" + "isen"
subprocess.check_output(cmd, shell=True)
```

On obtient alors bien le certificat de l'utilisateur délivré par notre AC :



L'utilisateur peut alors télécharger un fichier .zip depuis la page *success.html*.



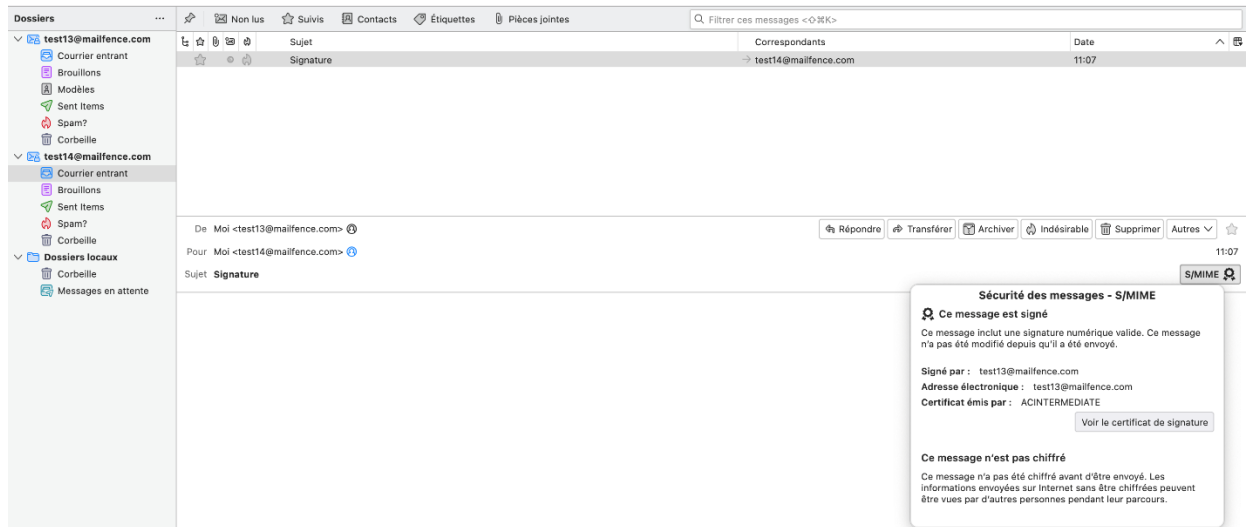
Ce fichier ZIP contient le certificat demandé, la clé privée, le certificat de l'ACI et celui de l'ACR.

Vous trouverez le script qui permet l'envoi du fichier .zip en annexe.

Thomas JAXEL
Hugo SAVY

7. Test de la signature de nos mails avec Thunderbird puis coté destinataire

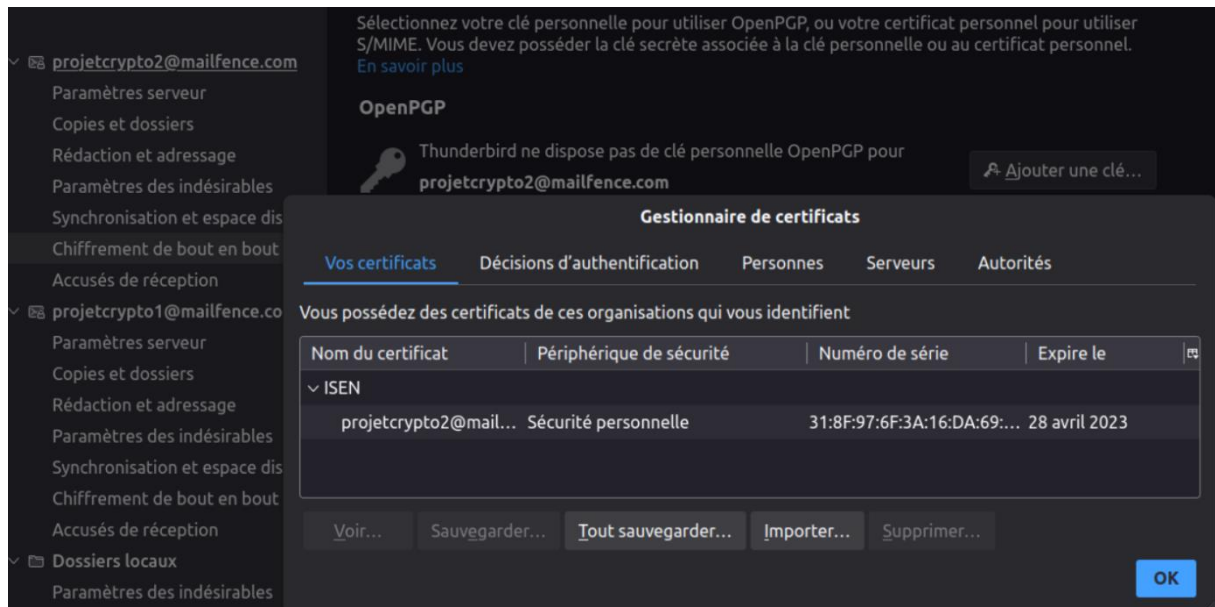
La signature de nos mails fonctionne :



8. Test avec des certificats non valides

Avec des certificats non valides, nous n'avons jamais réussi à dépasser l'étape où l'on donne à *Thunderbird* le certificat fauté. Le logiciel refusait de manière systématique le certificat.

a. Validé expirés



Thomas JAXEL
Hugo SAVY

Certificat pour projetcrypto2@mailfence.com - Mozilla Thunderbird

projetcrypto2@mailfence.com

Nom du sujet

Adresse e-mail	projetcrypto2@mailfence.com
Nom courant	projetcrypto2@mailfence.com
Organisation	ISEN
Unité organisationnelle	M1
Pays	FR
État / Province	PACA
Localité	TOULON

Nom de l'émetteur

Pays	FR
État / Province	PACA
Localité	TOULON
Organisation	ISEN
Unité organisationnelle	M1
Nom courant	ACINTERMEDIATE

Validité

Pas avant	Sat, 29 Apr 2023 08:39:54 GMT
Pas après	Fri, 28 Apr 2023 08:39:54 GMT

Informations sur la clé publique

Sélectionnez votre clé personnelle pour utiliser OpenPGP, ou votre certificat personnel pour utiliser S/MIME. Vous devez posséder la clé secrète associée à la clé personnelle ou au certificat personnel.
[En savoir plus](#)

OpenPGP

Thunderbird ne dispose pas de clé personnelle OpenPGP pour projetcrypto2@mailfence.com

[Ajouter une clé...](#)

Utilisez le gestionnaire de clés OpenPGP pour consulter et gérer les clés publiques de vos correspondants, ainsi que l'ensemble des autres clés non répertoriées ci-dessus.

Thunderbird

Le gestionnaire de certificats ne trouve pas de certificat valide qui puisse être utilisé pour signer numériquement vos messages avec une adresse <projetcrypto2@mailfence.com>.

[OK](#)

[Sélectionner un certificat...](#) [Effacer](#)

Certificat personnel pour le chiffrement :

[Sélectionner...](#) [Effacer](#)

Thomas JAXEL
Hugo SAVY

b. Mauvais key usage

Certificat pour projetcrypto2@mailfence.com - Mozilla Thunderbird

projetcrypto2@mailfence.com Paramètres des comptes X Certificat pour projetcrypto2@mailfence.com X

Algorithme	RSA
Taille de la clé	2048
Exposant	65537
Module	D6:F5:48:9C:BB:DF:5C:40:6A:83:E1:9C:F8:2E:D3:84:05:A6:1A:E7:0B:C8:47:28...

Divers

Numéro de série	10:76:19:D1:9A:24:01:16:AF:71:77:84:19:A0:E7:E3:BA:59:41:08
Algorithme de signature	ECDSA with SHA-256
Version	3
Télécharger	PEM (cert) PEM (chain)

Empreintes numériques

SHA-256	56:C6:EE:A4:3E:3A:59:11:5E:15:D5:83:FD:8E:1E:5D:EA:30:5C:70:03:23:68:52:...
SHA-1	0B:40:7A:89:3E:5C:2C:F7:C4:55:2E:FE:ED:9A:10:AF:DF:75:45:2D

Utilisations étendues de la clé

Usages	Server Authentication
--------	-----------------------

Identifiant de clé du sujet

ID de clé	72:D0:E2:2F:36:D4:1F:12:6E:A2:BF:45:A6:0B:66:33:4C:0B:C4:09
-----------	---

projetcrypto2@mailfence.com Paramètres serveur Copies et dossiers Rédaction et adressage Paramètres des indésirables Synchronisation et espace disque Chiffrement de bout en bout Accusés de réception

projetcrypto1@mailfence.com Paramètres serveur Copies et dossiers Rédaction et adressage Paramètres des indésirables Synchronisation et espace disque Chiffrement de bout en bout Accusés de réception

Dossiers locaux

Sélectionnez votre clé personnelle pour utiliser OpenPGP, ou votre certificat personnel pour utiliser S/MIME. Vous devez posséder la clé secrète associée à la clé personnelle ou au certificat personnel. [En savoir plus](#)

OpenPGP

Thunderbird ne dispose pas de clé personnelle OpenPGP pour projetcrypto2@mailfence.com [Ajouter une clé...](#)

Utilisez le gestionnaire de clés OpenPGP pour consulter et gérer les clés publiques de vos correspondants, ainsi que l'ensemble des autres clés non répertoriées ci-dessus.

Thunderbird

Le gestionnaire de certificats ne trouve pas de certificat valide qui puisse être utilisé pour signer numériquement vos messages avec une adresse <projetcrypto2@mailfence.com>.

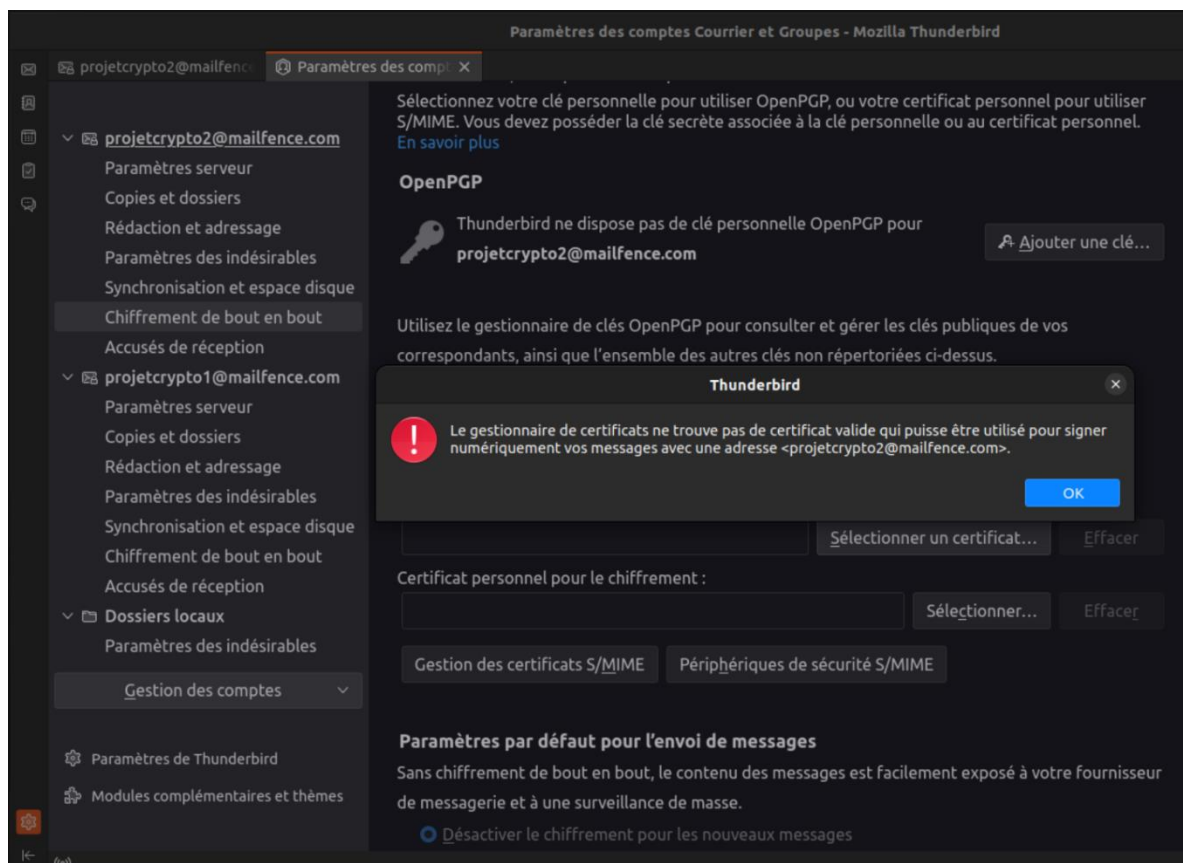
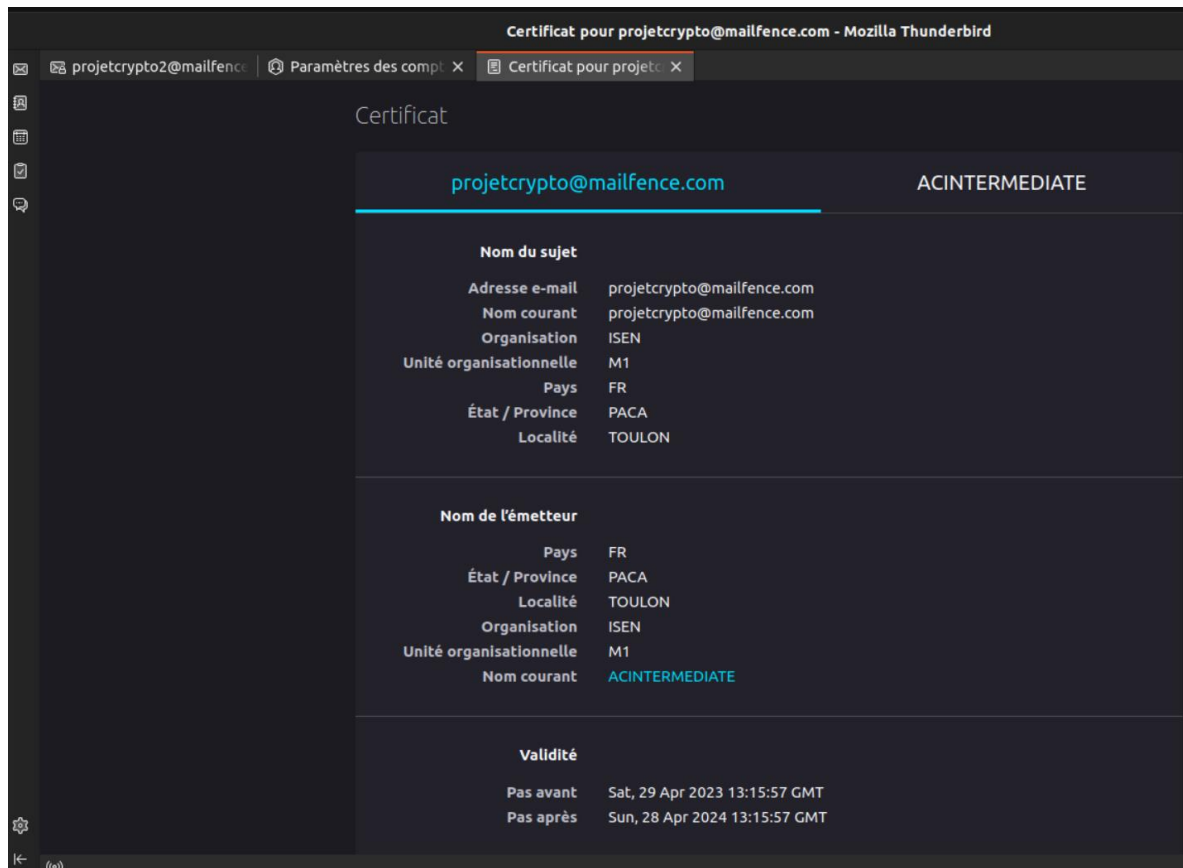
[Sélectionner un certificat...](#) [Effacer](#)

Certificat personnel pour le chiffrement :

[Sélectionner...](#) [Effacer](#)

Thomas JAXEL
Hugo SAVY

c. Faute dans le subject



- d. Clé différente de celle de l'ACI

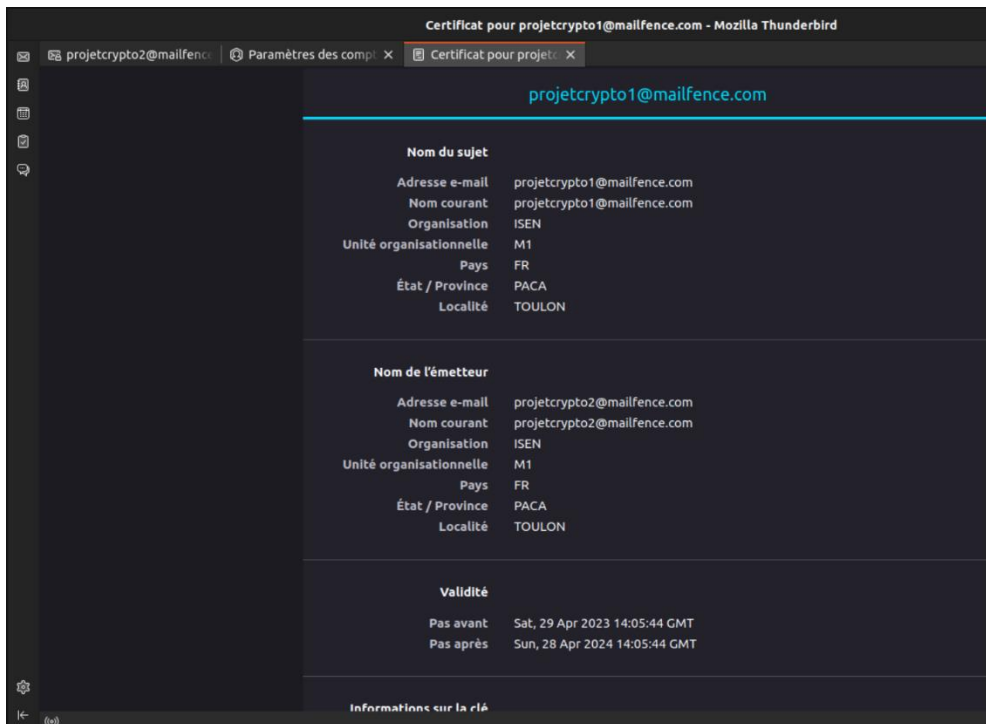
[illegible]

Ici le cas est différent puisqu'il n'est pas possible de créer un certificat avec une nouvelle clé et le certificat de l'ACI. L'erreur est avant Thunderbird.

Thomas JAXEL
Hugo SAVY

9. Attaque avec fausse AC

Pour réaliser cette attaque, on crée un certificat à partir de la clé privée et du certificat de projetcrypto2@mailfence.com.

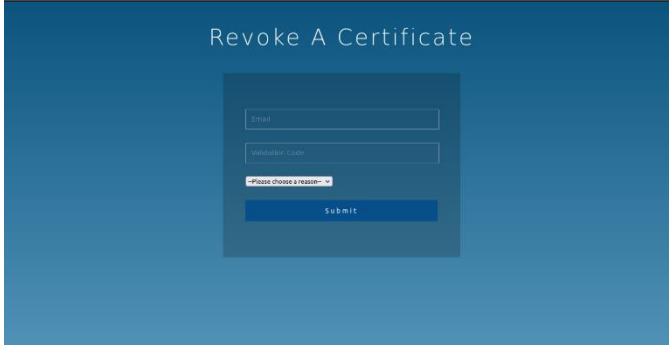


```
hugo@HP-HugoS: ~/PycharmProjects/cryptographie
hugo@HP-HugoS:~/PycharmProjects/cryptographie$ openssl x509 -noout -purpose -in projetcrypto1@mailfence.com.crt
Certificate purposes:
SSL client : Yes
SSL client CA : No
SSL server : Yes
SSL server CA : No
Netscape SSL server : Yes
Netscape SSL server CA : No
S/MIME signing : Yes
S/MIME signing CA : No
S/MIME encryption : Yes
S/MIME encryption CA : No
CRL signing : Yes
CRL signing CA : No
Any Purpose : Yes
Any Purpose CA : Yes
OCSP helper : Yes
OCSP helper CA : No
Time Stamp signing : No
Time Stamp signing CA : No
```

Comme on peut le voir avec la capture ci-dessus, le certificat créé par notre fausse AC ne fonctionnera jamais sur *Thunderbird* car le certificat utilisé pour créer notre fausse AC était celui d'un user, or notre vraie AC délivre des certificats avec des droits restreints afin de rendre impossible ce genre d'attaque. On pourrait modifier notre vraie AC pour rendre cette attaque possible mais nous n'en voyons pas l'intérêt. De ce fait, Thunderbird refuse de prendre ce certificat, car il ne dispose pas des droits CA.

10. Création du service de révocation de certificat

Pour notre service de révocation, nous avons mis en place un jeu de trois pages html permettant de rendre ce service plus user-friendly. Les pages sont : *revoke.html*, *revoke_success.html* et *revoke_error.html*



Revoke.html

Sur la page de révocation l'utilisateur est invité à saisir son adresse mail ainsi que le code unique qu'il a reçu lors de la création de son certificat. Ce code est important car il nous permet d'authentifier l'utilisateur et donc l'authenticité de la demande. Là encore, ce champ n'est pas bypassable car nous avons mis en place des mesures de protection (les mêmes que pour la page *verification.html*).

Sur cette page l'utilisateur doit préciser les raisons de cette révocation, ce qui nous permettra par la suite de préciser cette raison à notre serveur OCSP.

11. Création d'un serveur OCSP

Afin de créer notre propre serveur OCSP nous avons utilisé les commandes "openssl ocspp".

De plus, il a fallu mettre à jour nos certificats pour que ceux-ci soient connus de notre OCSP.

On démarre le serveur OCSP avec la commande ci-jointe :

```
openssl ocspp -index index.txt -port 8888 -rsigner ocspp.crt -rkey ocspp.key -CA  
../ACI/intermediate_ca.crt -text -out log.txt
```

Et on envoie la requête de création au serveur OCSP avec la commande suivante :

```
openssl ocspp -CAfile chain.pem -issuer chain.pem -cert " + cert + " -text -url http://localhost:8888
```

Enfin on envoie la requête de révocation de l'OCSP avec la commande suivante :

```
openssl ca -keyfile ACI/intermediate_ca.key -cert ACI/intermediate_ca.crt -revoke " + cert + " -  
crl_reason " + reason
```

Cependant, cette partie ne fonctionne pas de manière optimale. Pour plus d'information, merci de lire la partie concernée dans la section « difficultés rencontrées »

Les difficultés rencontrées

1. Création du CSR

Lors de la création de notre CSR, nous avons rencontré des problèmes de validité avec Thunderbird. Cela était causé par le « common name » qui était mal rempli par l'utilisateur lors de nos tests. Pour mettre fin à cette erreur nous avons donc enlevé la possibilité à l'utilisateur de remplir ce champ. Le « common name » est alors rempli par le traitement coté serveur avec l'adresse mail entrée par l'utilisateur.

2. Création du serveur OCSP

Lors de la création nous avons passé beaucoup de temps sur la compression du fonctionnement de *subprocess* pour pouvoir afficher les réponses du serveur sur notre terminal. Cela n'était pas tâche aisée car le terminal était occupé par notre serveur.

Lors de l'envoi d'une requête de création de certificat à notre serveur OCSP, nous nous heurtons à des messages signifiant que notre autorité de certification racine n'est pas reconnue.

```
Response Verify Failure
8014551872:error:27FFF070:OCSP routines:CRYPTO_internal:root ca not trusted:/AppleInternal/Library/BuildRoots/97f6331a-ba75-11ed-a4bc-863efbba880d/Library/C
aches/com.apple.xbs/Sources/LibressL/LibressL-3.3/crypto/ocsp/ocsp_vfy.c:168:
certs/test13@mailfence.com.crt: unknown
This Update: May 1 13:01:36 2023 GMT
thomas@MacBook-Pro-M1 /Users/thomas/PycharmProjects/cryptographie>
```

Nous n'avons jamais réussi à dépasser cette étape. Nous ignorons pourquoi notre « *root ca* » n'est pas trust par notre serveur OCSP.

L'OCSP ne fonctionne donc pas.

Cependant, en théorie si cette erreur est corrigée, le projet serait fini car la partie révocation semble fonctionnelle puisque le certificat révoqué est bien écrit dans la liste des certificats révoqués.

Les améliorations possibles

1. La vérification du code de vérification

La sécurité du code de vérification nous assurant qu'il s'agit bien l'utilisateur en question qui demande la création d'un certificat pourrait être renforcé en rajoutant par exemple un nombre limité de tentatives lors de la saisie du code dans la page de vérification pour éviter le brut-force.

2. Ajouter du HTTPS

À des fins de sécurité, la mise en place du HTTPS sur nos serveurs semble tout indiqué et serait en grand plus pour notre AC.

Thomas JAXEL
Hugo SAVY

Les ressources utilisées

<https://www.openssl.org/docs/man1.0.2/man1/openssl-req.html>

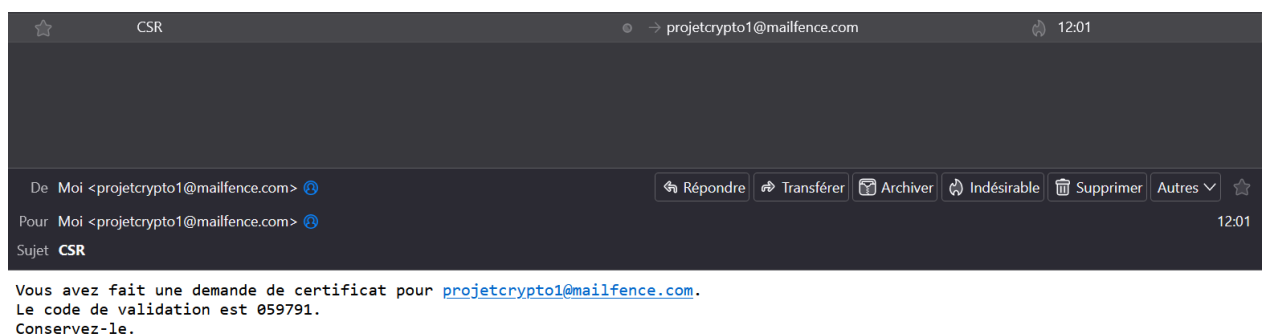
<https://www.tutorialspoint.com/How-to-create-a-zip-file-using-Python>

https://www.golinuxcloud.com/revoke-certificate-generate-crl-openssl/#OpenSSL_revoke_certificate_-_Introduction

<https://chat.openai.com/>

Annexes :

3. Mail avec code de vérification



Non lus : 0 Total : 4

4. Vérification de la CSR

```
# Vérification du CSR
csr_file = cn + ".csr"
cmd = "openssl req -noout -subject -in {}".format(csr_file)
subject_line = subprocess.check_output(cmd, shell=True).decode().strip()

if user_information == subject_line or user_information2 == subject_line:
    print("CSR correct")
else:
    print("Erreur(s) dans le CSR")
    return render_template('error.html')
```

5. Création et envoi du .zip

```
def download():  
    key_name = liste_info[7] + ".key"  
    certificate_name = liste_info[7] + ".crt"  
    archive_name = "key_and_certificate.zip"  
  
    # Création de l'archive contenant le certificat de l'utilisateur, sa paire de clé, l'ACR, l'ACI  
    with zipfile.ZipFile(archive_name, mode='w') as myzip:  
        myzip.write(certificate_name, certificate_name)  
        myzip.write(key_name, key_name)  
        myzip.write("ACR/root_ca.crt", "root_ca.crt")  
        myzip.write("ACI/intermediate_ca.crt", "intermediate_ca.crt")  
        myzip.close()  
  
    print("ZIP envoyé")  
  
    return send_file(archive_name, as_attachment=True)
```