



Master 1 : CRYPTIS Informatique

Rapport du projet de Sécurité TIC

Réalisé par :

Jean DE BONFILS

Sarra JLASSI

Version du
20 avril 2021

Table des matières

1	Organisation de l'autorité de certification	2
1.1	Autorité de certification Root	2
1.2	Autorité de certification intermédiaire	4
1.3	Génération de certificats	6
1.4	Timestamp Authority	7
2	Webservice	8
2.1	Fonction de création d'attestation	8
2.2	Fonction de vérification d'attestation	9
2.3	Le timestamp webservice	10
3	Client et relais TLS/TCP	10
4	Guide d'utilisation	11
4.1	Lancement du relais et des webservices	11
4.2	Programmes coté client et exemples	12
5	Analyse des risques	15

1 Organisation de l'autorité de certification

Voici tout d'abord le schéma représentant la chaîne de confiance de Certifplus que nous avons décidé d'implémenter :

Chaîne de confiance de certifplus

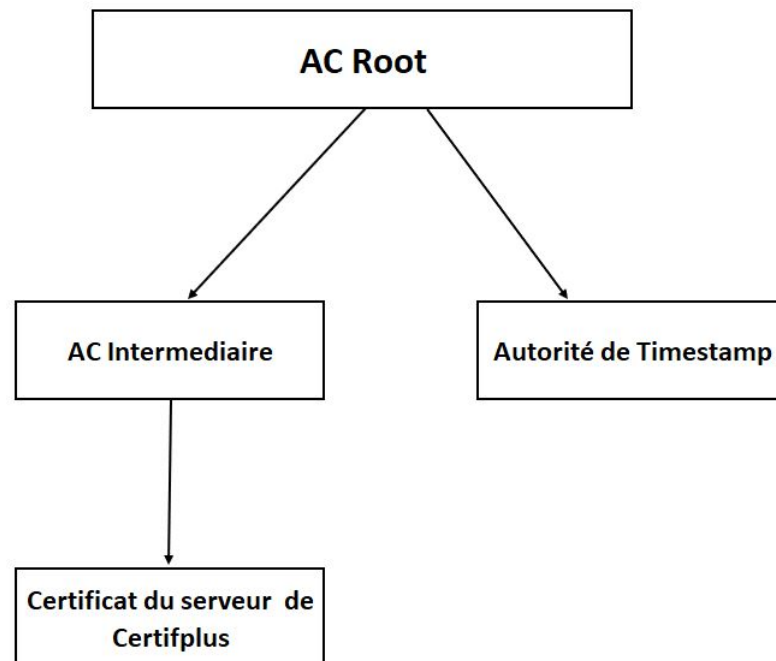


FIGURE 1

1.1 Autorité de certification Root

Tout d'abord, nous avons créé une **AC (Autorité de certification) Root** permettant de créer, plus tard, l'**AC intermédiaire** ainsi que le **Timestamp Authority (TSA)**. Cette **AC Root** est en haut de la hiérarchie de confiance et possède un certificat racine censé être utilisé par les clients pour vérifier l'authenticité d'un serveur qui dépend d'un AC intermédiaire qui dépend à son tour directement ou indirectement de l'AC Root.

Voici les commandes saisies pour créer l'AC Root :

```
mkdir ca $$ cd ca

#Creation de dossier dans lequel seront place les differents elements de l AC
mkdir certs crl newcerts private

#Restriction des droits
chmod 700 private

touch index.txt
echo 1000 > serial

#Creation du fichier de config pour l'AC
touch ca-root.cnf
.
.

#Creation de la clef utilisant les courbes elliptiques + chiffrement avec aes de cette dernière
openssl ecparam -genkey -name prime256v1 -outform PEM | openssl ec -aes256 -out private/ca-root.key.pem
#La passphrase: Certiplus-root-CA87000

chmod 400 private/ca-root.key.pem

#Creation du certificat racine à partir de la clé privé du ca
openssl req -config ca-root.cnf -key private/ca-root.key.pem -new -x509 -days 7300 -sha256 -extensions
v3_ca -out certs/ca-root.cert.pem

chmod 444 certs/ca-root.cert.pem

#Verification du certificat racine
openssl x509 -noout -text -in certs/ca-root.cert.pem
```

FIGURE 2 – Commande pour créer le CA Root

Tout d’abord, on crée les dossiers dans lesquels sont stockés les différents éléments de l’AC. On crée ensuite le fichier de configuration de l’AC (**ca-root.cnf**). Ce fichier de configuration reprend principalement le fichier **openssl.cnf** de base avec quelques parties modifiées. Notamment, on y indique les chemins vers les différents éléments de l’AC Root (le chemin vers le certificat, la clé privée... etc). Une partie correspondant à la configuration du TSA a aussi été rajouté (**voir I.4 Timestamp Authority**). On génère la clé privée basée sur les courbes elliptiques grâce à **ecparam**, puis on la chiffre grâce à l’algorithme de chiffrement symétrique AES256 et une passphrase. Cette clé privée nous permet, finalement, d’auto signer le certificat que l’on génère grâce à **openssl**. Enfin on vérifie le certificat obtenu grâce à la dernière commande.

Voici ce qui, entre autres, a été ajouté ou modifié au fichier **openssl.cnf** de base (d’autres choses ont été ajouté notamment pour la configuration du TSA) :

```
#Definition des chemins vers les différents elements du CA
[ CA_default ]
dir                = .
certs              = $dir/certs
crl_dir            = $dir/crl
new_certs_dir      = $dir/newcerts
database           = $dir/index.txt
serial             = $dir/serial
RANDFILE           = $dir/private/.rand
private_key        = $dir/private/ca-root.key.pem
certificate        = $dir/certs/ca-root.cert.pem
.
.
#Certaines informations sont obligatoire lors de la creation d un AC intermediaire
[ policy_strict ]
countryName        = match
stateOrProvinceName = optional
organizationName   = match
organizationalUnitName = optional
.
.
[ req ]
default_bits       = 4096
distinguished_name = req_distinguished_name
string_mask        = utf8only
default_md         = sha256
.
.
[ v3_intermediate_ca ]
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always,issuer
#Pathlen:0 indique que les CA intermediaires ne pourront pas créer eux même de CA
basicConstraints = critical, CA:true, pathlen:0
keyUsage = critical, digitalSignature, cRLSign, keyCertSign
```

FIGURE 3 – Extrait du fichier ca-root.cnf

1.2 Autorité de certification intermédiaire

Après avoir créé l'AC Root nous pouvons donc désormais créer l'AC Inter-médiaire qui va dépendre de ce dernier :

```
#Dans le repertoire de l'AC root on créer un repertoire dédié à l'AC intermediaire
mkdir intermediate $$ cd intermediate

#Creation de dossier dans lequel seront place les differents elements de l AC
mkdir certs crt csr newcerts private
chmod 700 private
touch index.txt
echo 1000 > serial
echo 1000 > crlnumber

#Creation du fichier de config de l'AC intermediaire
touch ca-intermediate.cnf
.
.

#Creation de la cle utilisant les courbes elliptiques + chiffrement avec aes de cette dernière
openssl ecparam -genkey -name prime256v1 -outform PEM | openssl ec -aes256 -out private/ca-intermediate.key.pem
#La passphrase que j'ai mis: Certiplus-intermediate-CA87000

cd ../

#Creation du certificate signing request
openssl req -config intermediate/ca-intermediate.cnf -new -sha256 -key intermediate/private/ca-intermediate.key.pem
-out intermediate/csr/intermediate.csr.pem

#On utilise le fichier de config de root-ca.cnf pour signer (avec la clé du ca root) le CSR créé précédement
openssl ca -config ca-root.cnf -extensions v3_intermediate_ca -days 3650 -notext -md sha256 -in intermediate/csr/intermediate.csr.pem
-out intermediate/certs/intermediate.cert.pem
#Puis on saisie la passphrase de ca-root.key.pem (Certiplus-root-CA87000)
#Sign the certificate? [y/n]: y
#Commit ? [y/n]: y

chmod 444 intermediate/certs/intermediate.cert.pem

#Verification du certificat intermediaire
openssl x509 -noout -text -in intermediate/certs/intermediate.cert.pem

#Verification du certificat intermediaire par rapport au certificat root
openssl verify -CAfile certs/ca-root.cert.pem intermediate/certs/intermediate.cert.pem
#Le resultat doit etre : intermediate/certs/intermediate.cert.pem: OK

#Creation de la chaine de certificat:
cat intermediate/certs/intermediate.cert.pem certs/ca-root.cert.pem > intermediate/certs/ca-chain.cert.pem
```

FIGURE 4 – Commande pour créer le CA intermédiaire

Dans le répertoire de l'AC Root, on crée un répertoire dédié à l'AC intermédiaire dans lequel seront stockés les différents éléments du CA ainsi que le fichier de configuration du CA **ca-intermediate.cnf**, ce fichier ressemble plus ou moins à la configuration de **ca-root.cnf** avec certaines parties en moins et avec les chemins vers les éléments du CA modifiés.

De même que pour le CA Root, on crée la clé privée utilisant les courbes elliptiques et on la chiffre avec **AES**. Grâce à cette clé privée, on va pouvoir créer un "**Certificate Signing request**" que l'on va soumettre à signature à l'AC Root. Une fois le **CSR** généré, on crée le certificat intermédiaire qui va être signé par l'AC Root grâce à sa clé privée. (Les chemins vers les clés privées des AC sont indiqués dans leurs fichiers de configuration). On saisit, ensuite, le passphrase pour déchiffrer la clé du CA Root qui avait été chiffré à l'aide d'AES. Enfin, on vérifie que le certificat généré est bien correct puis on crée la chaîne de certificat en concaténant les deux certificats.

1.3 Génération de certificats

Maintenant que nous avons le CA intermédiaire nous allons pouvoir l'utiliser pour générer un certificat pour le serveur de Certifplus. En effet, il est recommandé de créer un CA Root qui s'occupe seulement de créer des CA intermédiaires qui eux génèrent les certificats pour les serveurs.

```
#Génération de la clé utilisant les courbes élliptiques
openssl ecparam -out intermediate/private/certifplus.key.pem -name prime256v1 -genkey

cd intermediate/

#Création du CSR à partir de la clé privé générée et de la configuration du CA intermédiaire (ca-intermediate.cnf)
openssl req -config ca-intermediate.cnf -key private/certifplus.key.pem -new -sha256 -out csr/certifplus.csr.pem

#Création du certificat à partir du csr
openssl ca -config ca-intermediate.cnf -extensions server_cert -days 375 -notext -md sha256 -in csr/certifplus.csr.pem
-out certs/certifplus.cert.pem
#Dans intermediate/index.txt on doit voir : 1000 unknown ... /CN=certifplus

#Vérification du certificat :
openssl x509 -noout -text -in certs/certifplus.cert.pem
#Résultat(Issuer = ca-intermediate,Subject = serveur Certifplus)
Issuer: C = FR, ST = France, O = Certifplus, OU = CA-Certifplus, CN = CA intermédiaire de Certifplus,
emailAddress = jean.de-bonfils@etu.unilim.fr
Validity
Not Before: Apr  7 13:40:14 2021 GMT
Not After : Apr 17 13:40:14 2022 GMT
Subject: C = FR, ST = France, L = Limoges, O = Certifplus, OU = Attestation Unit, CN = Unite d attestation de Certifplus,
emailAddress = jeandebonfils@gmail.com

#On vérifie que le certificat créé a une chaîne de confiance valide
openssl verify -CAfile certs/ca-chain.cert.pem certs/certifplus.cert.pem
#Résultat:
certs/certifplus.cert.pem: OK
```

FIGURE 5 – Commande pour créer le certificat du serveur

De la même façon que précédemment, on va générer la clé, le "Certificate Signing Request" qui nous permet de générer le certificat du serveur signé par l'AC Intermédiaire. Lors de la création du certificat, l'option "**-extensions server-cert**" doit être ajoutée. La clé générée au départ permettant de créer le "Certificate Signing Request" ne sera par contre pas chiffrée avec AES comme précédemment avec les CA Root et intermédiaire. On peut vérifier, ensuite, le certificat généré notamment avec la commande **openssl x509 -noout -text -in certs/certifplus.cert.pem**. On doit pouvoir clairement voir l' "Issuer" qui correspond à l'AC intermédiaire et le "Subject" qui correspond au serveur de Certifplus. Enfin, on vérifie, grâce à la chaîne de certificat créée précédemment, que le certificat généré a une chaîne de confiance valide.

1.4 Timestamp Authority

Le TSA "Timestamp Authority" crée dépend directement du CA Root et non pas du CA intermédiaire. Pour la création de celui-ci, certains éléments doivent être ajoutés au fichier de configuration du CA Root :

```
oid_section                                = new_oids

[ new_oids ]
tsa_policy1                                = 1.2.3.4.1
tsa_policy2                                = 1.2.3.4.5.6
tsa_policy3                                = 1.2.3.4.5.7
.
.
.

[ tsa ]
default_tsa                                = tsa_config1

[ tsa_config1 ]
dir                                         = .                                # TSA root directory
serial                                     = tsa_serial
signer_cert                               = tsa.crt
certs                                     = tsa-ca-chain.cert.pem
signer_key                                 = private/tsa.key
default_policy                             = tsa_policy1
signer_digest                             = sha256
#other_policies                           = tsa_policy2,tsa_policy3
digests                                   = sha256,sha384,sha512
accuracy                                  = secs:1,millisecs:500,microsecs:100
ordering                                  = yes
tsa_name                                  = yes
ess_cert_id_chain                         = yes
ess_cert_id_alg                           = sha256

[ tsa_ext ]
authorityKeyIdentifier                     = keyid:always
basicConstraints                           = critical,CA:false
extendedKeyUsage                           = critical,timeStamping
keyUsage                                   = critical,nonRepudiation
subjectKeyIdentifier                       = hash
```

FIGURE 6 – Ajout à réaliser dans le fichier ca-root.cnf

De même que pour le CA intermédiaire, on crée un répertoire dédié au TSA. On crée ensuite la clé privée ainsi que le "Certificate Signing Request". On soumet ensuite cette requête au CA Root pour générer le certificat du TSA. (L'option "-extensions tsa_ext" doit être indiquée dans la commande). Enfin on crée le fichier représentant la chaîne de confiance.


```
#Création d'un certificate signing request et de clé avec RSA
openssl req -new -newkey rsa:2048 -subj "/C=FR/O=Certiplus/OU=TSA Certifplus/CN=TSA Certifplus"
-keyout TSA/private/tsa.key -out TSA/tsa.csr
passphrase : Certiplus-TSA-CA87000

#Génération du certificat à partir du csr
openssl ca -config ca-root.cnf -in TSA/tsa.csr -out TSA/tsa.crt -extensions tsa_ext -days 365

#Création de la chaîne de certificat
cat TSA/tsa.crt certs/ca-root.cert.pem > TSA/tsa-ca-chain.cert.pem
```

FIGURE 7 – Commande pour créer le TSA

La création de **TimeStamp request** et **timestamp reply** se fait ensuite via un **Webservices**(voir II.3) auquel on passe le fichier à horodater.

2 Webservice

2.1 Fonction de création d'attestation

Dans la fonction `creation_attestation()` associée à la route `/creations` :

- On récupère, tout d'abord, les informations de l'étudiant saisies par l'utilisateur. (l'identité : nom et prénom et l'intitulé de la certification) et on les écrit dans le fichier **infosEtudiant.txt** stocké dans un répertoire temporaire qui est créé à la réception d'une requête de création et qui est détruit avant que l'attestation soit retournée à l'utilisateur. Ce dossier permet notamment de stocker tous les fichiers temporaires.
- Ensuite, on signe le bloc d'informations avec la clé privée de la société de certification **CertifPlus** à l'aide de la commande :
`subprocess.run("openssl dgst sha256 sign ../CA/intermediate/private/-certifplus.key.pem out "+signInfosEtuPath+" "+infosEtuPath, shell= True);`
- On génère du texte à combiner dans l'image finale à l'aide du webservice ChartAPI fournie par Google à l'aide d'une requête réalisée avec l'outil CURL. L'image est stockée au format PNG dans **texte.png** et redimensionnée avec ImageMagick .
- On crée le fichier du QRcode **qrcode.png** contenant la signature du bloc d'informations dans le fichier **signature.sigsha256** et on l'ajoute à l'image

finale **attestation.png**.

- On concatène le bloc d'informations et on rajoute des caractères afin d'avoir une longueur de 64.
- On récupère, ensuite, les données du le timestamp reply de la part du TimeStamp authority de Certiplus et on les ajoute aux données de l'étudiant.
- Enfin, on cache ces données par steganography dans l'image et on renvoie le résultat **attestation** à l'utilisateur .

2.2 *Fonction de vérification d'attestation*

Dans la fonction **vérification_attestation()** associée à la route **/verification** :

- On récupère ici tout d'abord l'attestation envoyée par l'utilisateur .
- On récupère par la suite les informations contenues(la signature) dans le QRCode dans le fichier **Signqrcode**.
- De même que pour la création d'attestation, un répertoire temporaire est créé le temps de la vérification de l'attestation.
- On récupère également les informations de l'étudiant grâce à la fonction **enleverBourage()** qui permet d'enlever les caractères de bourrage insérés lors de l'exécution de la fonction **création attestation()** et le timestamp contenus dans l'image **imageAtt**. Les 64 premiers caractères retrouvés contiennent les informations de l'étudiant et du bourrage.
- Finalement, on fait appel à la fonction **Timestamp_Verify()** afin de vérifier la validité du timestamp du fichier **verifInfosEtu.txt**. Cette fonction utilise le certificat du TSA créé précédemment ainsi que le certificat de l'AC Root pour vérifier le timestamp. La fonction **Signature_Verify()** afin de vérifier la signature du fichier grâce à la clé publique de Certifplus. Si le timestamp et la signature sont corrects, alors l'attestation est vérifiée.

2.3 Le timestamp webservice

Pour la génération des timestamps, nous avons créé un webservice (**0.0.0.0:8060/gen-TimeStamp**) en python qui récupère le fichier à horodater via une requête curl du client et qui retourne le timestamp au format .tsr . D'abord, le webservice génère un timestamp request à partir duquel il génère le timestamp. On utilise en paramètre de ces deux commandes **-config ca-root.cnf** pour indiquer à openssl d'utiliser le fichier de configuration du CA Root dans lequel on a mis la configuration du TSA.

```

10 @route('/genTimeStamp', method='POST')
11 def create_time_stamp():
12
13     #Recuperation du fichier à timestamp
14     fileToTS = request.files.get('file')
15     fileToTS.save("out/fileTMP")
16     print("Génération d'un timestamp pour "+fileToTS.filename)
17
18     #Generation du timestamp
19     subprocess.run("openssl ts -query -config ../ca-root.cnf -data out/fileTMP -out out/request.tsq -sha384",shell=True)
20     subprocess.run("openssl ts -reply -passin pass:Certipius-TSA-CA87000 -config ../ca-root.cnf -queryfile out/request.tsq -out out/response.tsr")
21
22     response.set_header('Content-type', 'application/timestamp-query')
23     descripteur_fichier=open("out/response.tsr",'rb')
24     tsaResp=descripteur_fichier.read()
25     descripteur_fichier.close()
26
27     #On supprime les fichiers TMP
28     subprocess.run("rm out/*",shell=True)
29
30     return tsaResp
31
32 if __name__ == '__main__':
33     print("Time Stamp Webservice : En attente d'une connexion")
34     run(host='0.0.0.0',port=8060,debug=True)
35

```

FIGURE 8 – Code du Timestamp Webservice

Lors de la vérification de l'attestation, le serveur n'a pas besoin du webservice pour vérifier le timestamp. Le serveur a seulement besoin du certificat du CA Root ainsi que du certificat du TSA. De la même façon que pour **freettsa.org**, la création du timestamp nécessite un webservice mais la vérification de ce dernier n'en nécessite pas tant que le serveur possède les certificats du TSA et de l'AC Root.

3 Client et relais TLS/TCP

Le relais **TLS/TCP** se fait entre le client, le serveur frontal et le serveur applicatif. Le serveur frontal reçoit les requêtes des clients et établit une connexion sécurisée grâce au protocole TLS et au certificat du serveur de Certifplus. Une fois que la connexion entre le client et le serveur frontal est établie, la requête du

client est transmise au serveur applicatif qui renvoie à son tour une réponse au serveur frontal qui se charge de la transmettre au client par la connexion TLS établie précédemment.

La mise en place du relais se fait via une simple commande **socat openssl-listen** en passant en paramètre le certificat du serveur de Certifplus ainsi que sa clé privée. Le client peut de son côté vérifier l'authenticité du serveur frontal grâce au certificat racine de l'autorité de certification Root qu'il possède et qui contient notamment la clé publique. Le certificat du serveur fait partie de la même chaîne de certificats que l'AC Root (**voir fig 1**) .

Côté client, nous avons écrit 4 scripts :

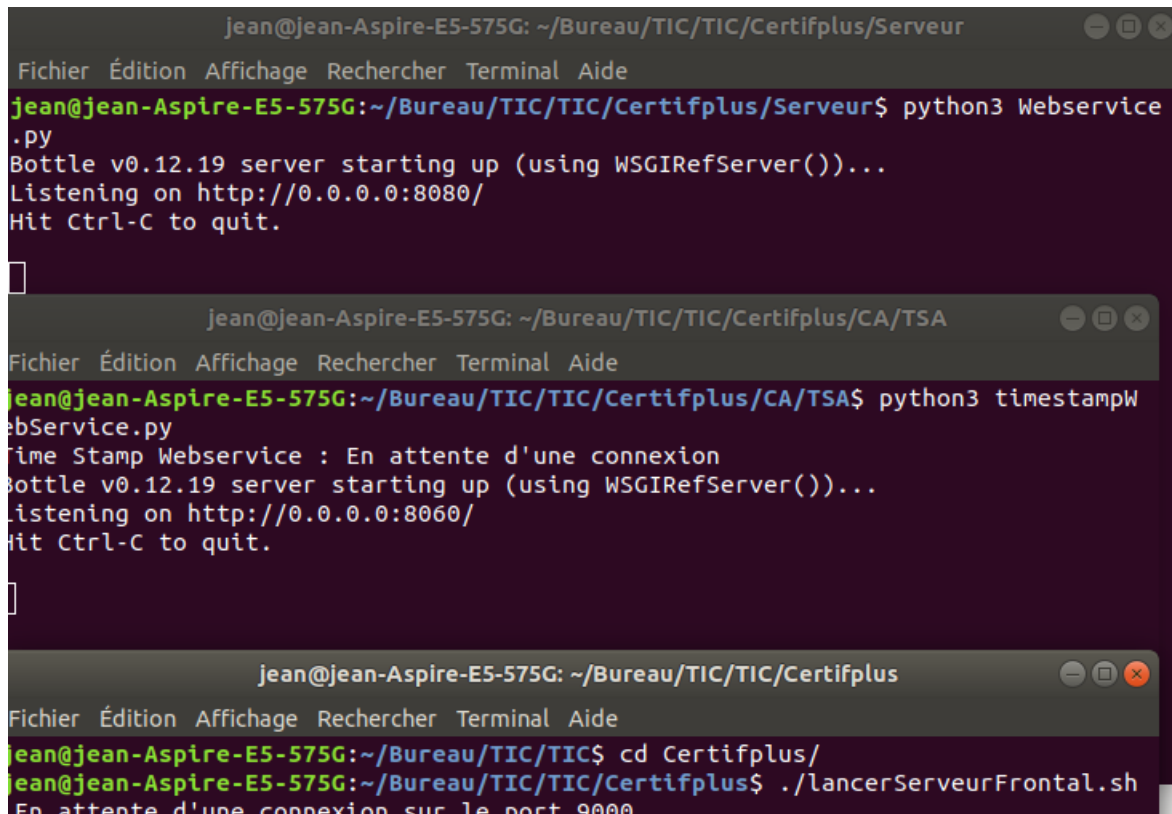
- Un script Python3 pour générer une attestation à l'aide d'une interface graphique.
- Un script Bash permettant de générer une attestation en ligne de commande
- Un script Python3 générant une interface graphique et qui permet de vérifier une attestation.
- Un script Bash permettant de vérifier une attestation en ligne de commande

En effet, l'utilisation des programmes côté client doit être simple car ils seraient censés être utilisés par n'importe qui voulant créer ou vérifier une attestation (d'où l'interface graphique).

4 Guide d'utilisation

4.1 Lancement du relais et des webservices

Avant de pouvoir utiliser les clients, les différents services doivent être allumés. D'une part on lance le webservice permettant de générer et créer une attestation. Ce webservice se trouve dans le dossier **Certifplus/serveur**. D'autre part on lance le timestamp webservice qui se trouve dans **Certifplus/CA/TSA** et qui va être utilisé par le premier webservice pour obtenir des timestamps. Enfin, on lance le relais TLS/TCP servant de serveur frontal et qui se trouve dans **Certifplus/** :



```
jean@jean-Aspire-E5-575G: ~/Bureau/TIC/TIC/Certifplus/Serveur
Fichier Édition Affichage Rechercher Terminal Aide
jean@jean-Aspire-E5-575G:~/Bureau/TIC/TIC/Certifplus/Serveur$ python3 Webservice
.py
Bottle v0.12.19 server starting up (using WSGIRefServer())...
Listening on http://0.0.0.0:8080/
Hit Ctrl-C to quit.

jean@jean-Aspire-E5-575G: ~/Bureau/TIC/TIC/Certifplus/CA/TSA
Fichier Édition Affichage Rechercher Terminal Aide
jean@jean-Aspire-E5-575G:~/Bureau/TIC/TIC/Certifplus/CA/TSA$ python3 timestampW
ebService.py
Time Stamp Webservice : En attente d'une connexion
Bottle v0.12.19 server starting up (using WSGIRefServer())...
Listening on http://0.0.0.0:8060/
Hit Ctrl-C to quit.

jean@jean-Aspire-E5-575G: ~/Bureau/TIC/TIC/Certifplus
Fichier Édition Affichage Rechercher Terminal Aide
jean@jean-Aspire-E5-575G:~/Bureau/TIC/TIC$ cd Certifplus/
jean@jean-Aspire-E5-575G:~/Bureau/TIC/TIC/Certifplus$ ./lancerServeurFrontal.sh
En attente d'une connexion sur le port 9000
```

FIGURE 9 – Commandes pour lancer les différents services

4.2 Programmes coté client et exemples

Les programmes côté client sont prêts à être utilisés, ils se trouvent dans le dossier **client/**.

On peut créer une attestation à l'aide d'une interface graphique en lançant **python3 creerAttestationGUI.py**. Un formulaire s'affiche alors permettant de saisir le nom, le prénom et l'intitulé. Ensuite, l'utilisateur peut choisir un répertoire dans lequel enregistrer l'attestation. L'attestation reçue aura pour nom "**attestation _nom_de_famille.png**". Enfin si tout s'est bien passé un message de confirmation s'affiche.

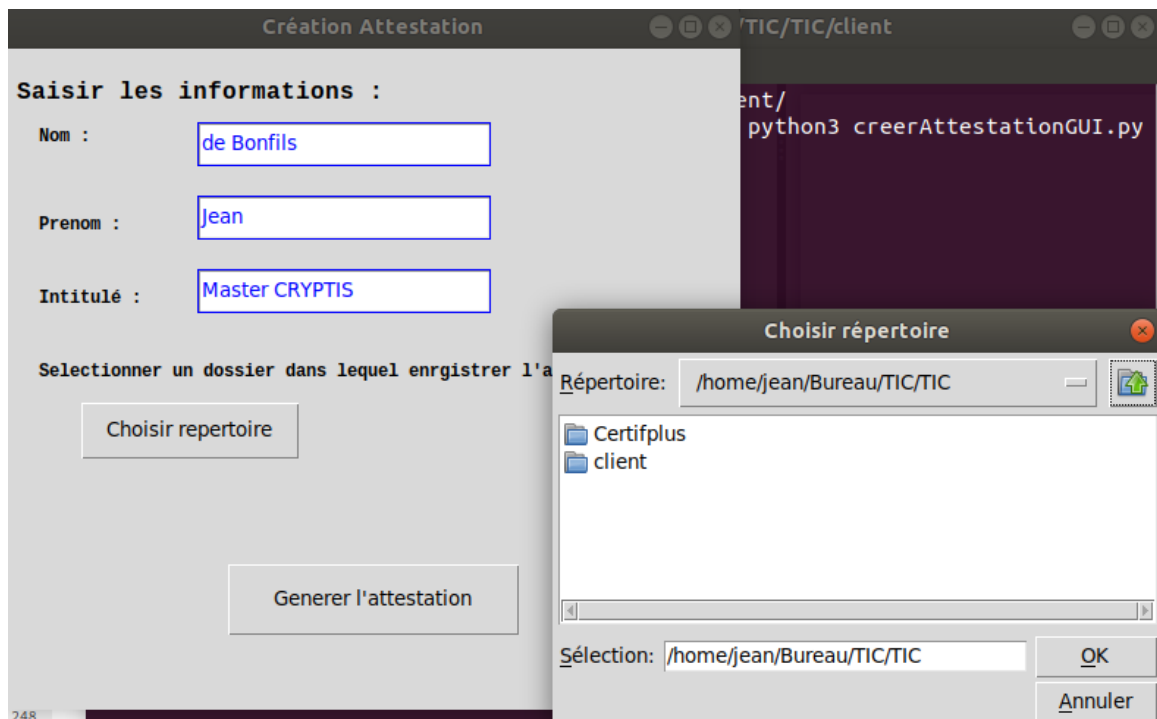


FIGURE 10 – Interface graphique permettant de créer une attestation

L'utilisateur peut aussi créer une attestation directement en ligne de commande : `./creerAttestation.sh -o chemin/ou/enregistrer/attestation` Le programme demande alors à l'utilisateur de saisir les informations de l'étudiant. Voici une attestation générée par les programmes précédents :

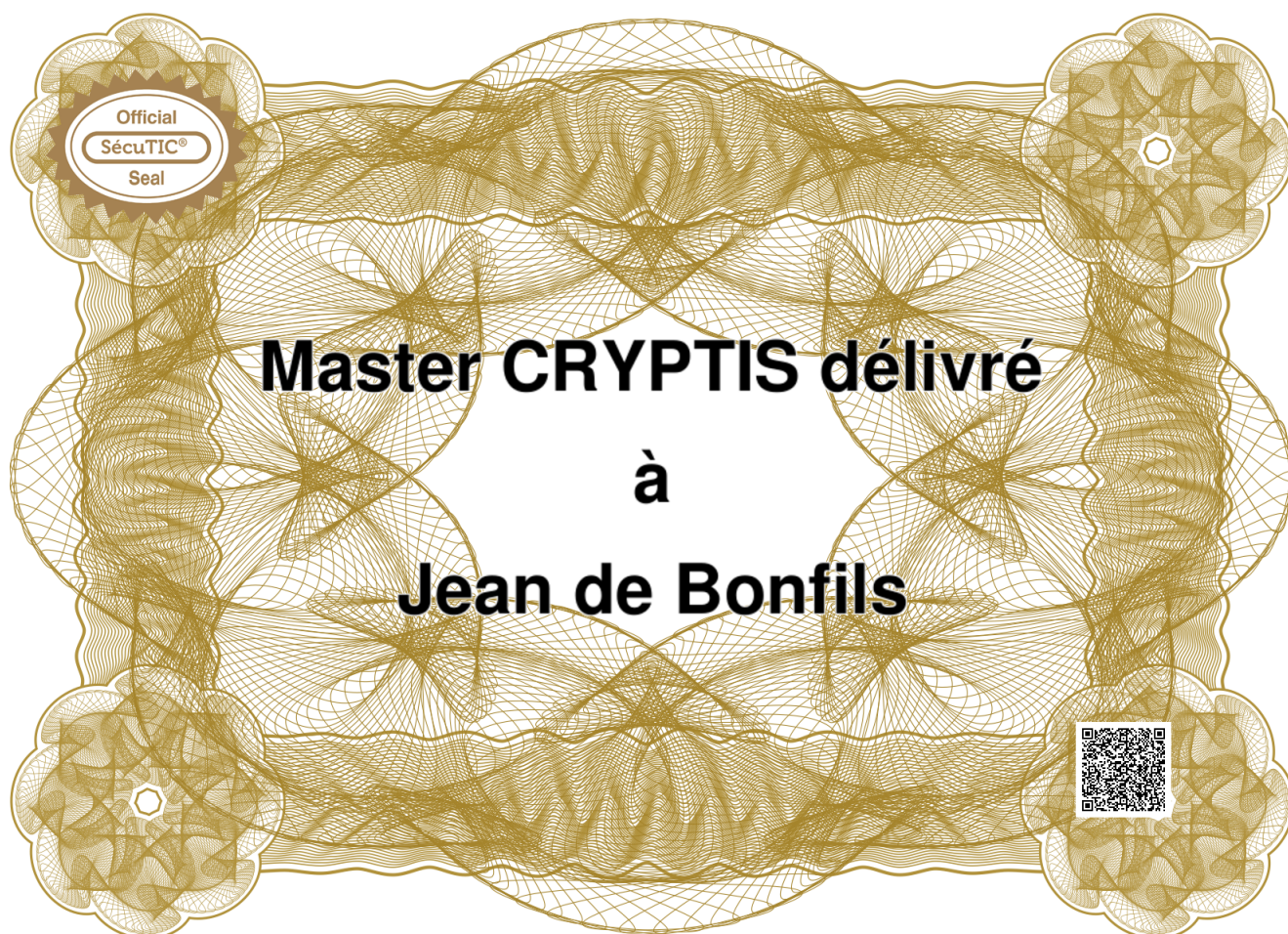


FIGURE 11 – Exemple d’une attestation générée

Un utilisateur peut ensuite vérifier une attestation en lançant **python3 verifierAttestationGUI.py**. Une fenêtre s’affiche alors, l’utilisateur peut sélectionner un fichier via une boîte de dialogue. Le fichier est envoyé au webservice qui vérifie l’attestation et qui renvoie un message au client attestant ou non de la validité du fichier.

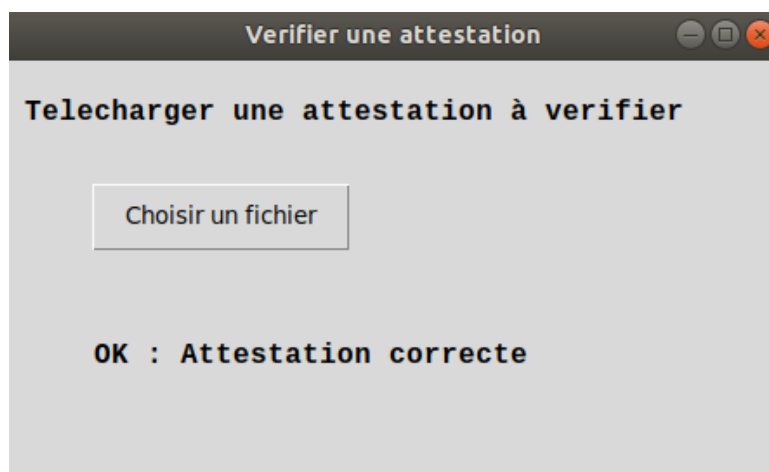


FIGURE 12 – Vérification d'une attestation valide

Si on passe une attestation erronée ou un autre fichier, "**ERREUR : Attestation incorrecte**" s'affiche à la place. Une attestation peut aussi être vérifiée en ligne de commande en y indiquant le chemin du fichier. Voici un exemple de vérification en ligne de commande d'une attestation **erronée** :

```
jean@jean-Aspire-E5-575G:~/Bureau/TIC/TIC/client$ ./verifierAttestation.sh attestationErrone.png
Envoie de la requete de verification...
ERREUR : Attestation incorrecte jean@jean-Aspire-E5-575G:~/Bureau/TIC/TIC/client$
```

FIGURE 13 – Vérification d'une attestation erronée en ligne de commande

Pour plus d'informations sur l'organisation du rendu merci de regarder le fichier "installation.txt".

5 Analyse des risques

Dans cette partie , on va identifier les biens à protéger par la société CertifPlus en tant qu'actifs primaires et actifs secondaires et dégager les menaces qui pèsent sur ces biens , tout en précisant les relations qui existent entre eux.

Les actifs primaires correspondent à un un besoin de l'entreprise. Ils sont généralement représentés par un service ou par les données nécessaires au fonctionnement d'un service. **Les actifs secondaires** quant à elles représentent les moyens nécessaires à la réalisation des besoins fonctionnels décrits par les actifs primaires .

- **Les données de l'utilisateur** sont considérés comme des **actifs primaires** . En effet , ces données doivent être échangées et ne doivent pas être modifiées . Pour ce faire, il est nécessaire de signer les informations par les clés privées .
- **Les clés privées des autorités de certification** réalisant la signature peuvent être considérées comme des **actifs primaires**. Elles sont absolument indispensables aux besoins de l'entreprise. Le vol ou la perte de la clé privée du CA Root serait très grave puisque cela rendrait toute la chaîne de certification inutilisable(CA intermédiaire, TSA, certificats associés). Le vol de cette dernière pourrait permettre à n'importe qui de générer des certificats à partir du CA de Certifplus et donc usurper l'identité de Certifplus. Bien que ce scénario soit très grave, il est peu probable. De plus, les clés privées ont été préalablement chiffrés avec l'algorithme de chiffrement symétrique AES256. Il est donc en plus nécessaire d'avoir la passphrase pour pouvoir utiliser ces clés (cela réduisant le risque). De plus, les dossiers contenant les informations sensibles telles que les clés privées possèdent des droits d'accès restreints pour réduire le risque d'un tel scénario.
- **Les passphrases** permettant d'accéder aux clés privées de l'AC Root et intermédiaire sont considérées comme des **actifs secondaires** . Elles sont un moyen nécessaire à l'accès aux clés privées de l'AC Root (actifs primaires) . Le vol de la passphrase permettant d'accéder à la clé privée de l'AC Root serait grave car cela causerait, comme pour la perte des clés privées elle-même, une invalidation de la chaîne de certificats. Les clés privées sont les éléments les plus importants et donc, par extension, les passphrases d'accès à ces clés le sont aussi. Bien que la perte des clés, au niveau de l'AC intermédiaire, soit grave, il suffirait de révoquer l'AC Root et de le recréer, cela ne détruirait donc pas toute la chaîne de confiance (Le TSA par exemple ne serait pas impacté). L'accès aux passphrases doit donc être restreint et connu par un minimum de personnes pour réduire les risques. *"Le secret le mieux gardé est celui que l'on garde pour soi."*
- **Les certificats** sont considérés comme des **actifs primaires** . La perte du certificat du serveur de Certifplus ne serait pas grave. En effet, on pourrait simplement en régénérer un nouveau en quelques commandes. Cependant, la perte du certificat de l'AC Root serait plus problématique. Ce certificat est utilisé par les clients pour vérifier l'authenticité du serveur de Certifplus au moment de la création de la communication TLS. Il existe

aussi un risque de fraude du certificat. Si un utilisateur malveillant réussit à retrouver les clés. Cependant ce risque, bien que grave, a très peu de chances d'arriver.

- **Les programmes côté client et les webservices** sont considérés comme des **actifs secondaires** . En effet, même si la perte de ces programmes entraînerait une indisponibilité des services de création et de vérification d'attestation, cela ne serait que temporaire et ne constitue pas une menace grave. Dans ce cas-là, le mieux serait de garder une copie de ces programmes au cas où un utilisateur les supprimerait malencontreusement. Cette menace, bien qu'elle soit probable, n'est pas grave et est donc considéré comme acceptable , mais le fonctionnement de ces programmes pourra être corrompu si un utilisateur malveillant y a accès , ainsi qu'aux clés et certificats (actifs primaires).
- **Le serveur frontal** est considéré comme un **actif secondaire**. Ce serveur contrairement aux serveurs applicatifs est directement accessible depuis internet, il est donc exposé directement aux attaques de l'extérieur. Cependant, l'indisponibilité temporaire de celui-ci n'est pas grave et ne causerait pas de dommages ou de menaces aux autres biens. Cependant, une mauvaise configuration de celui-ci pourrait avoir des répercussions sur le serveur applicatif. L'utilisation d'un proxy d'une tierce partie place toute la triade Confidentialité, Intégrité et Disponibilité entre les mains de cette tierce partie. Dans notre cas le serveur frontal est géré par Certifplus la triade Confidentialité, Intégrité et Disponibilité reste donc interne à l'entreprise.
- **Le serveur applicatif** est considéré comme un **actif secondaire** . Celui-ci est assez peu exposé aux risques. En effet le rôle du serveur frontal est justement de protéger le serveur applicatif contre toute action malveillante. La probabilité d'un dommage sur le serveur applicatif est donc en grande partie liée à la configuration du serveur frontale.
- **Le service d'horodatage** est considéré comme un **actif primaire**. En effet , ce service permet de garantir la non-répudiation ainsi que l'intégrité dans le temps et pour réaliser ce besoin fonctionnel, nous avons besoin du **fichier de requête timestamp** qui assure le critère de preuve et considéré comme **actif secondaire** . Pour cela , ce service est compromis si on perd le fichier de requête timestamp .