Rapport - Sécurité TIC.
2020-2021

Faculté des Sciences et Techniques
Master 1 - Maths. CRYPTIS

Authentification, Web sécurisé & Stéganographie

A l'attention de :
BONNEFOI P-F.

Rédigé par :
PIARD A.
JACQUET R.

Table des matières

Introduction

Le but du projet était de mettre en œuvre un procédé de diffusion électronique sécurisé d'attestation de réussite pour la société de certification **CertifPlus**.
Nous devions dans ce travail avoir recours à un **WebService** dans lequel un étudiant pouvait demander son attestation de réussite et également dans lequel un éventuel employeur pouvait vérifier cette même attestation. Il fallait donc, sous-jacent à ces deux possibilités, permettre la création d'une attestation en fonction du nom de l'étudiant et y ajouter ce qui serait nécessaire à sa vérification par l'employeur. C'est ici que la stéganographie intervient majoritairement, avec la mise en place d'un **QRcode** contenant des informations chiffrées.

1 Analyse du programme

Notre code est composé de trois fichiers PYTHON. On trouve les fichiers *serveur_web.py*, *scripts.py* et *stegano.py*.

1.1 Fonctions notables

Ce fichier python contient l'ensemble des fonctions qui nous seront utiles lors de ce projet. On y trouve par exemple une fonction permettant d'ajouter un QRcode à une image, ou encore une fonction permettant d'obtenir le timestamp.
Ce fichier de fonctions évite de surcharger notre fichier **serveur_web.py** et se compose de commandes données dans le sujet et de nos propres fonctions.

1.1.1 get_timestamp()

```
1 def get_timestamp():
2     commande = subprocess.Popen("openssl ts -query -data texte.png -no_nonce
3     "-sha512 -cert -out texte.tsq", shell=True,stdout=subprocess.PIPE)
4     (resultat, ignorer) = commande.communicate()
5     if not resultat:
6         return "Erreur sur le timestamp"
7     else:
8         return resultat
```

Cette fonction nous donne le timestamp, obtenu à partir du texte présent sur (la future) attestation. Le timestamp est stocké dans le fichier *texte.tsq*, obtenu à partir de la commande de la ligne 2, trouvée sur le site www.freetsa.org.

1.1.2 creation_signature(contenu)

```
1 def creation_signature(contenu):
2     commande = subprocess.Popen("openssl dgst -sha256 -sign ecc.ca.key.pem %s
3     "> timestamp.bin"%contenu, shell=True,stdout=subprocess.PIPE)
4     (resultat, ignorer) = commande.communicate()
5     if not resultat:
6         return "Erreur sur la signature"
7     else:
8         return resultat
```

Cette fonction prend en paramètre un fichier pour le signer à l'aide de la clé privée créée avec notre autorité de certification, basée sur les courbes elliptiques. Pour effectuer cette signature, l'empreinte du fichier placé en paramètre est faite à l'aide de l'algorithme SHA256, puis cette empreinte est chiffrée avec la clé privée évoquée précédemment, qui est une clé privée de l'algorithme de chiffrement ECDSA.
La signature est stockée dans le fichier *timestamp.bin*.

1.1.3 resizeqr(qrcode)

Cette fonction était nécessaire pour plusieurs raisons. Lors de l'insertion du QRcode sur l'image, par la commande spécifiée dans le sujet et utilisée dans la fonction make_qr_on_img(data), notre QRcode sortait de l'image. Il n'était visible que lorsque nous changions les coordonnées de la commande par 700 et 500 par exemple, au lieu de 1418 et 934.

```
composite -geometry +1418+934 qrcode.png combinaison.png attestation.png
```

Nous avons donc fait quelques essais et nous avons également demandé de l'aide aux responsables de ce projet. Il s'est avéré que la commande était bonne, malgré nos doutes dû à l'impossibilité de voir le QRcode sur l'image. Au final, notre problème venait de notre image fond_attestation, récupérée sur le lien du projet, qui n'était pas au bon format.
Ensuite, nous avons vu dans le sujet la commande

```
attestation.crop((1418,934, 1418 + 210, 934+210))
```

qui nous indique que le QRcode que l'on récupère est au format 210×210. La fonction RESIZEQR(QRCODE) permet donc de redimensionner, aux bonnes dimensions, le QRcode qu'on souhaite insérer sur l'attestation.

1.1.4 make_qr_on_img(data)

```
def make_qr_on_img(data):
    fichier = open(data,"rb")
    temp = ""
    for caractere in fichier:
        temp += str(caractere)
    fichier.close()
    nom_fichier = "qrcode.png"
    qr = qrcode.make(temp)
    qr.save(nom_fichier, scale=2)
    resizeqr("qrcode.png")
    commande = subprocess.Popen("composite -gravity center texte.png
    "fond_attestation.png combinaison.png", shell=True,stdout=subprocess.PIPE)
    (resultat1, ignorer1) = commande.communicate()
    commande2 = subprocess.Popen("composite -geometry +1418+934 qrcode.png
    "combinaison.png attestation.png", shell=True,stdout=subprocess.PIPE)
    (resultat2, ignorer2) = commande2.communicate()
    if not resultat2 or not resultat1:
        return "Erreur sur la création du QR sur l'image"
    else:
        return "OK!"
```

Le paramètre de cette fonction est un fichier qu'on ouvre pour en lire les bytes.

On crée alors un QRcode qui contient les bytes du fichier ouvert précédemment (sous forme de chaîne de caractères). On centre ensuite le texte sur la future attestation puis on y intègre le QRcode, créé au préalable, aux coordonnées 1418×934.

1.1.5 verif_sig(signature)

```
def verif_sig(signature):
    commande1 = subprocess.Popen("openssl dgst -sha256 -verify pubkey.pem -
    "signature %s texte.tsq"%signature, shell=True,stdout=subprocess.PIPE)
    (resultat1, ignorer1) = commande1.communicate()
    print(resultat1)
    if not resultat1:
        return "Erreur sur le déchiffrement"
    else:
        return "La signature est correcte"
```

Cette commande prend en paramètre un fichier stockant la signature à vérifier. Dans notre cas il s'agira de *timestamp.bin*. La commande utilisée dans cette fonction permet de vérifier la signature d'un fichier signé par ECDSA avec la clé privée citée dernièrement et sa clé publique associée **pubkey.pem**.

1.2 Modifications hors protocole

Dans le SERVEUR WEB nous avons apporté une modification simple mais majeure pour permettre d'afficher les caractères **utf-8** sur l'attestation. Il s'agit de,

```
contenu_identité = request.forms.identite
contenu_intitulé_certification = request.forms.intitule_certif
```

qui remplace :

```
contenu_identité = request.forms.get('identite')
contenu_intitulé_certification = request.forms.get('intitule_certif')
```

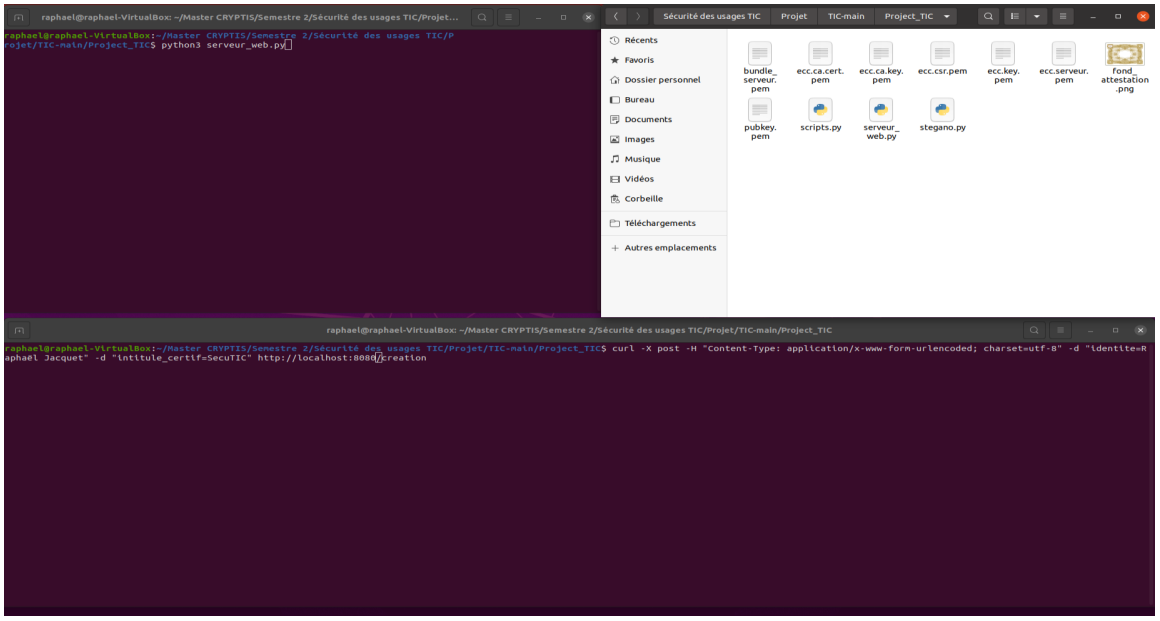
Notamment, pour la vérification de la signature nous avons utilisé la commande de la fonction VERIF_SIG(SIGNATURE) mais en travaux pratiques nous avions utilisé une autre méthode. Cette dernière est la comparaison de la signature qu'on retire du QRcode avec la création de la signature à partir des informations présentes sur le QRcode, avec le même algorithme pour l'empreinte et la même clé de chiffrement.

2 Exécution

Dans cette section vous trouverez les commandes nécessaires à la bonnes exécution de notre code, ainsi que des captures d'écrans pour en afficher le rendu.
L'autorité de certification ainsi que ses fichiers en découlant ont été obtenus à partir du TP5 sur les courbes elliptiques et possèdent donc les mêmes appellations. Pour ce qui est de la clé publique, nous l'avons obtenue à partir de la commande : openssl ec -in ecc.ca.key.pem -pubout -out pubkey.pem.

2.1 Initialisation

Avant de créer une attestation, notre fichier est donc composé de ces documents : et on peut voir que les terminaux sont prêts à être exécutés.

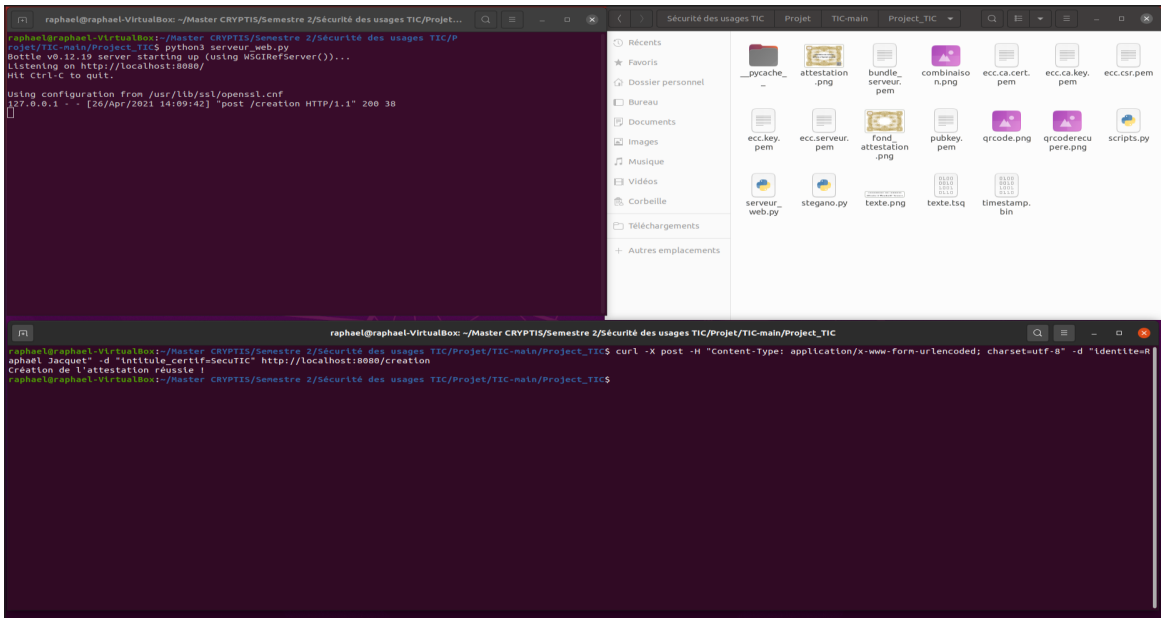


Pour exécuter le programme il faut écrire python3 serveur_web.py dans un terminal. Il faut également avoir ouvert un autre terminal, tous les deux depuis le dossier présenté ci-dessus. Ce dernier terminal est utilisé pour la communication avec le serveur. C'est dans celui-ci que l'on demandera ou bien la création, ou bien la vérification d'une attestation.

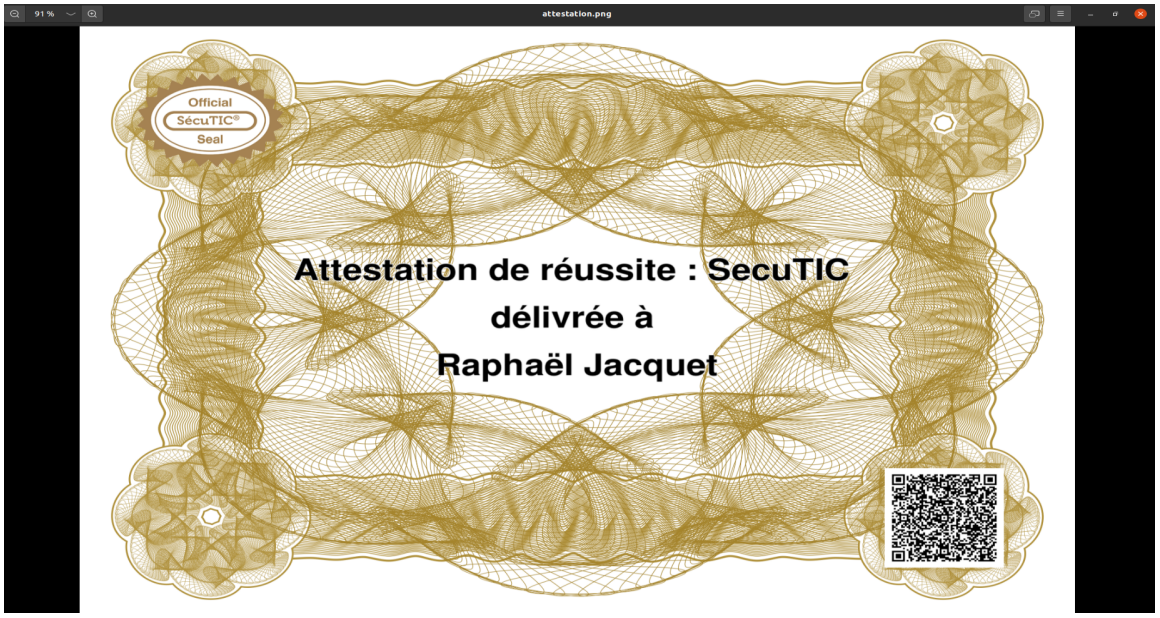
2.2 Création d'une attestation

Pour demander la création d'une attestation avec comme nom Raphaël Jacquet, pour voir la gestion des caractères **utf-8**, et comme intitulé de certification SecuTIC, la commande à écrire dans le terminal est, sans retour à la ligne :

```
curl -X post -H "Content-Type: application/x-www-form-urlencoded; charset=utf-8"
 -d "identite=Raphaël Jacquet" -d "intitule\_certif=SecuTIC"
http://localhost:8080/creation
```



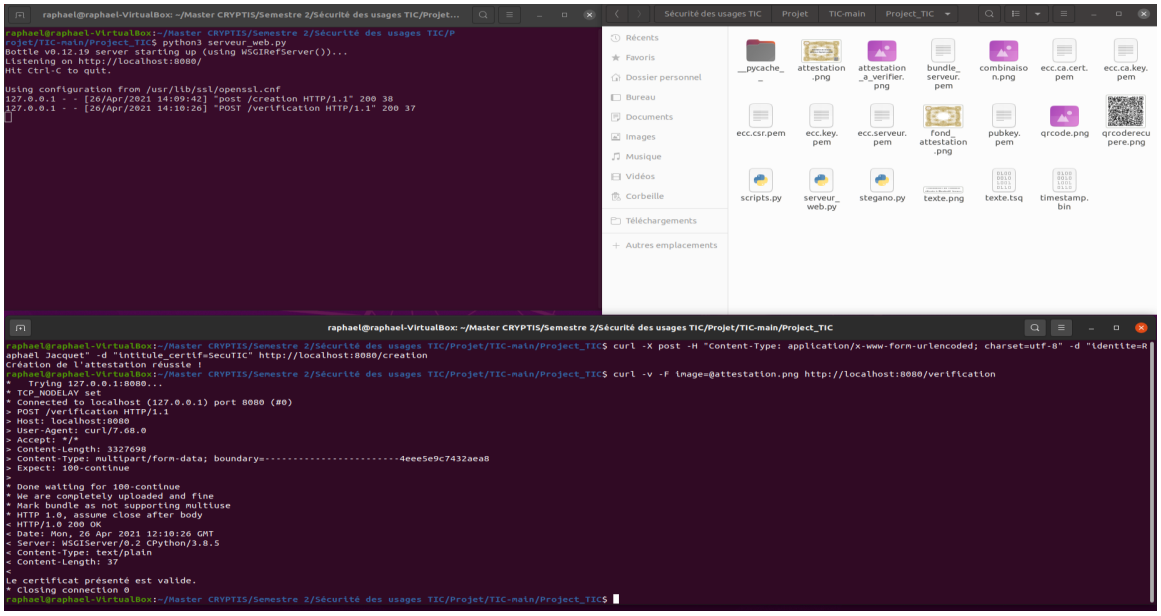
On observe que le programme renvoie "Création de l'attestation réussie!" dans le terminal une fois que l'attestation a été créée.
Une fois cette étape franchie, vous trouverez dans votre dossier le fichier attestation.png, qui n'est autre que le suivant :



2.3 Vérification d'une attestation

Enfin, pour demander la vérification d'une attestation, on écrira :

```
curl -v -F image=@attestation.png http://localhost:8080/verification
```



Cette dernière commande nous renverra "Le certificat présenté est valide" si la vérification de la signature contenue dans le QRcode est bien celle attendue. Si ce n'est pas le cas, vous observerez le message "L'attestation n'est pas valide".

3 Analyse de risques

Si l'on considère que deux serveurs sont présent, le serveur applicatif et le frontal, ce qui n'est pas notre cas, il y a quand même des risques que nous allons identifier. Faire deux "serveurs" en local (d'hôte 'localhost'), même s'ils communiquent sur deux ports différents n'est pas un gage de sécurité. Le problème est qu'avec une identification du port du serveur applicatif il est très simple de déjouer la sécurité de communication TLS entreprise par le serveur frontal. De plus, comme pour tout serveur ne possédant pas de couche (layer) anti dénis de service, il est fortement possible que le serveur s'écroule sous les demandes de certificats ou de vérifications. Pour palier à ce risque nous avons simplement implémenter une vérification du timestamp local et permit une seule opération du Serveur Web toutes les demi-secondes afin d'éviter une surcharge. Dans notre cas la communication sur le port 9000 n'a pas été concluante malgré la bonne exécution du lancement avec SOCAT. Le serveur frontal et applicatif sont donc fusionnés ce qui n'est clairement pas optimal en terme de sécurité sur les communications puisqu'elles sont directes. Pour garantir une meilleure productivité et éviter au serveur de travailler inutilement, nous avons implémenter une vérification du prénom utilisant les expressions régulières.

Nous pourrions résumer notre analyse de risque avec ce tableau :

Risque	**Contrôlé**	**Présent**
Attaque DDoS	X	
Communication directe		X
Contrôle des données entrantes	X	
Serveur applicatif non isolé		X
Sécurité de l'AC	X	
Authenticité de l'attestation	X	
Intégrité des fichiers sensibles grâce à la signature	X	

L'authenticité est bien vérifiée grâce à la signature du timestamp. Les fichiers sensibles sont sauvegardés sur le serveur, que l'on peut considéré comme sécurisé. Des menaces de pénétration système restent présente mais l'objectif de ce projet n'était pas de palier à ces failles. Nous avons cependant fait notre maximum pour protéger le serveur de plusieurs attaques possibles.

Conclusion

L'ensemble des consignes du sujet ont été satisfaites, mis à part les commandes inhérentes au lancement du serveur frontal. C'est donc selon nous un projet assez bien mené. Des petites implémentations ont été ajoutées aux consignes initiales pour permettre de sécuriser légèrement plus notre application, comme précisé lors de l'analyse de risques. Ce projet nous aura légèrement ouvert à la stéganographie, bien que le principe ne nous était pas inconnu. Ceci a étendu notre intérêt pour l'Unité d'Enseignement **Sécurité des Usages TIC** ainsi que les connaissances qu'il nous a apporté. Par ailleurs, l'utilisation du QRcode pour authentifier un document est une idée qui nous a plu. Le sujet de ce projet a été formateur, autant en sécurité car il a reprit tous les éléments que nous avions vu en TP, qu'en langage **Python** que nous avions déjà travaillé au semestre précédent. Pour un "mini projet" il y avait beaucoup d'éléments à assimiler, modifier et ajouter. Cela l'a rendu complexe donc instructif.
Finalement, nous sommes satisfaits du travail que nous vous rendons malgré notre problème au niveau du serveur, car nous avons ajoutés des fonctionnalités à notre programme et car l'objectif du projet a été accompli.