



Université  
de Limoges

## Master 1 Informatique

# Rapport de projet Sécurité des usages TIC

---

## Authentification, Web sécurisé & Stéganographie

---

Réalisé par :

JAMOUSSE Houcein

KALAI Mohamed Hedi

BARHOUMI Hadir

Enseignants :

Monsieur BONNEFOI Pierre-Francois

Monsieur CONCHON Emmanuel

# Table des matières

<b>1</b>	<b>Cadre général du projet</b>	<b>1</b>
<b>2</b>	<b>Mise en place de l'architecture</b>	<b>1</b>
2.1	Le programme serveur_web.py . . . . .	1
2.1.1	Fonction creation_attestation . . . . .	1
2.1.2	Fonction verification_attestation . . . . .	2
2.1.3	Code stéganographie . . . . .	4
2.2	Création des certificats . . . . .	4
2.2.1	Autorité de certification . . . . .	4
2.2.2	Serveur . . . . .	5
2.3	Connexion TLS : « TCP » avec socat . . . . .	6
<b>3</b>	<b>Exemple d'exécution</b>	<b>7</b>
3.1	Création de l'attestation . . . . .	7
3.2	Vérification de l'attestation . . . . .	7
<b>4</b>	<b>Analyse des risques</b>	<b>8</b>
4.1	Les actifs . . . . .	8
4.1.1	Les actifs primaires . . . . .	8
4.1.2	Les actifs secondaires . . . . .	8
4.2	Les menaces . . . . .	8
4.3	Les relations . . . . .	9
<b>5</b>	<b>Conclusion</b>	<b>9</b>

## Table des figures

1	Fonction creation_attestation partie1 . . . . .	1
2	Fonction creation_attestation partie2 . . . . .	1
3	Fonction creation_attestation partie3 . . . . .	2
4	Fonction creation_attestation partie4 . . . . .	2
5	Fonction creation_attestation partie5 . . . . .	2
6	Fonction creation_attestation partie6 . . . . .	2
7	Fonction verification_attestation partie1 . . . . .	3
8	Fonction verification_attestation partie2 . . . . .	3
9	Fonction verification_attestation partie3 . . . . .	3
10	Fonction verification_attestation partie4 . . . . .	4
11	Exemple : Certificat du serveur . . . . .	5
12	TCP avec Socat . . . . .	6
13	TLS Handshake pour la création d'attestation . . . . .	6
14	TLS Handshake pour la vérification d'attestation . . . . .	6
15	Création de l'attestation . . . . .	7
16	Affichage du serveur_web pour la création de l'attestation . . . . .	7
17	Vérification de l'attestation . . . . .	7
18	Affichage du serveur_web pour la vérification de l'attestation . . . . .	8

# 1 Cadre général du projet

Ce projet a pour objectif de mettre en place un système de diffusion électronique sécurisée d'attestations de réussite aux certifications délivrées par la société CertifPlus. Le système garantira l'authenticité des attestations sous forme d'image, grâce à une information visible comprenant le nom de la personne ayant obtenu la certification, le nom de la certification réussie et un QRcode contenant la signature de ces informations, ainsi qu'une information dissimulée par stéganographie dans l'image. Cette information contiendra les mêmes informations visibles de l'attestation ainsi que la date de délivrance garantie par un « timestamp » signé par l'autorité d'horodatage « [www.freetsa.org](http://www.freetsa.org) ».

Le système sera implémenté sous forme d'un Webservice ou d'une Application Web, utilisant le protocole HTTP et supportant deux types de requêtes : la récupération de l'attestation et la vérification de l'attestation. Le système sera conçu pour répondre aux besoins de la société CertifPlus et agira en tant qu'autorité de certification pour la délivrance de certificats nécessaires à la mise en place de ce service.

## 2 Mise en place de l'architecture

### 2.1 Le programme serveur\_web.py

Le serveur web possède deux routes :

- "/creation"
- "/verification"

#### 2.1.1 Fonction creation\_attestation

En gros, cette fonction permet de créer une attestation de réussite. Comment ? Elle récupère deux données via le formulaire : l'identité de l'étudiant et l'intitulé de la certification, concatène ces informations et ajoute des caractères si nécessaire pour atteindre une taille de 64 octets. Elle génère ensuite un timestamp signé par une autorité d'horodatage « [www.freetsa.org](http://www.freetsa.org) ».

```
#####
# Concaténation du nom et prénom de l'étudiant avec l'intitulé de la certification
infos = contenu_identité + contenu_intitulé_certification
if len(infos) < 64: # Ajout de caractères '@' pour atteindre une taille de 64 octets
    for i in range(64 - len(infos)):
        infos += "@"
with open("infos.txt", "wb") as f:
    f.write(infos.encode())
```

FIGURE 1 – Fonction creation\_attestation partie1

```
#####
#TimeStamp
command1 = subprocess.Popen("openssl ts -query -data infos.txt -no_nonce -sha256 -cert -out time.tsq", shell=True, stdout=subprocess.PIPE)
stdout, stderr = command1.communicate()
command2 = subprocess.Popen("curl -H \"Content-Type: application/timestamp-query\" --data-binary '@time.tsq' https://freetsa.org/tsr > time.tsr", shell=True, stdout=subprocess.PIPE)
stdout, stderr = command2.communicate()
command3 = subprocess.Popen('openssl base64 -in time.tsr -out time_64.tsr', shell=True, stdout=subprocess.PIPE)
stdout, stderr = command3.communicate()
with open("time_64.tsr", "rb") as f:
    timestamp = f.read()
```

FIGURE 2 – Fonction creation\_attestation partie2

Dans l'étape suivante, on génère un texte contenant le nom et prénom de l'étudiant, ensuite on utilise l'API Google Chart pour créer une image de ce texte. Ce texte est alors redimensionné à une taille de 1000x600 pixels en utilisant **ImageMagick**.

```
#####
#Texte image finale
texte_ligne = "Attestation de réussite|délivrée à " + contenu_identité
commande = "curl -o texte.png \"http://chart.apis.google.com/chart\" --data-urlencode \"chst=d_text_outline\" --data-urlencode \"chld=000000|56|\"
+ \"\"
command4 = subprocess.Popen(commande, shell=True, stdout=subprocess.PIPE)
stdout,stderr = command4.communicate()
#Redimensionnement avec ImageMagick de l'image texte.png
command5 = subprocess.Popen("mogrify -resize 1000x600 texte.png", shell=True, stdout=subprocess.PIPE)
stdout,stderr = command5.communicate()
```

FIGURE 3 – Fonction creation\_attestation partie3

Puis, on crée un QR code à partir de la signature du fichier **infos.txt** qu'on a créé dans la première étape. La bibliothèque QR Code est utilisée pour créer le code, qui est ensuite enregistré sous le nom **qrcode.png**.

```
#####
#QRCODE
#Signature du fichier infos.txt
command6 = subprocess.Popen("openssl dgst -sha256 -sign cle_CA.pem infos.txt | openssl base64 -out signature_64.tsr", shell=True, stdout=subprocess.PIPE)
stdout,stderr = command6.communicate()
with open('signature_64.tsr', 'rb') as f:
    signature = f.read()
signature=signature.decode()
nom_fichier = "qrcode.png"
qr = qrcode.make(signature)
qr.save(nom_fichier, scale=2)

#Redimensionnement avec ImageMagick de l'image qrcode.png
command7=subprocess.Popen("mogrify -resize 210x210 qrcode.png", shell=True, stdout=subprocess.PIPE)
stdout,stderr=command7.communicate()
```

FIGURE 4 – Fonction creation\_attestation partie4

On passe ensuite à combiner les images **texte.png**, **fond\_attestation.png** donnée et le QRCode pour créer l'image d'attestation finale.

```
#####
#Insertion de texte.png+fond_attestation.png+qrcode.png pour avoir l'image attestation.png
command8=subprocess.Popen("composite -gravity center texte.png fond_attestation.png fond_text.png", shell=True, stdout=subprocess.PIPE)
stdout,stderr=command8.communicate()
command9=subprocess.Popen("composite -geometry +1418+934 qrcode.png fond_text.png attestation.png", shell=True, stdout=subprocess.PIPE)
stdout,stderr=command9.communicate()
```

FIGURE 5 – Fonction creation\_attestation partie5

Finalement, les informations et le timestamp sont dissimulés dans l'image finale en utilisant la fonction **cacher** pour cacher les données dans les bits de poids faible des pixels. L'image finale est ensuite retournée en tant que contenu image/png.

```
#####
#Dissimulation des infos et de la timestamp dans l'image
image = Image.open("attestation.png")
cacher(image,infos+timestamp.decode())
image.save("attest_final.png")
response.set_header('Content-type', 'image/png')
with open('attest_final.png', 'rb') as f:
    contenu = f.read()
```

FIGURE 6 – Fonction creation\_attestation partie6

### 2.1.2 Fonction verification\_attestation

La première partie concerne l'extraction de contenu caché dans une image stéganographique. La fonction **request.files.get('image')** récupère l'image entrée par l'utilisateur, qui est ensuite sauvegardée sous le nom **verif\_attest.png**. La variable **msg** récupère le contenu caché, qui est ensuite découpé en deux parties : la variable **timestamp** qui contient les informations temporelles, et la variable **identite** qui contient les informations d'identification.

```

@route('/verification', method='POST')
def vérification_attestation():
    #####
    #extraction du contenu stegano
    contenu_image = request.files.get('image')
    contenu_image.save('verif_attest.png', overwrite=True)
    attest_a_verif = Image.open("verif_attest.png")
    msg = recuperer(attest_a_verif, taille)
    timestamp = msg[64:]
    identite = msg[:64]

```

FIGURE 7 – Fonction verification\_attestation partie1

La deuxième étape concerne ce qu'on vient d'extraire : le timestamp et l'identité. Premièrement, le timestamp est encodé en base64 et stocké dans un fichier appelé **time\_64.tsr**. Ensuite, on décode ce timestamp dans le fichier **time\_decode.tsr**. Le contenu de ce dernier fichier est lu et stocké dans un fichier appelé **timestamp.tsr**. Deuxièmement, l'identité est stockée dans un fichier appelé **infos1.txt**.

```

#####
#conversion de timestamp et identite
with open("time_64.tsr", "wb") as f:
    f.write(timestamp.encode())
command1 = subprocess.Popen('openssl base64 -d -in time_64.tsr -out time_decode.tsr', shell=True, stdout=subprocess.PIPE)
stdout, stderr = command1.communicate()
with open("time_decode.tsr", "rb") as f:
    timestamp = f.read()
with open("timestamp.tsr", "wb") as f:
    f.write(timestamp)
with open("infos1.txt", "wb") as f:
    f.write(identite.encode())

```

FIGURE 8 – Fonction verification\_attestation partie2

On passe ensuite à la vérification du timestamp. On génère une requête de timestamp à partir des données contenues dans le fichier **infos1.txt**. Ensuite, on utilise cette requête pour vérifier l'intégrité du timestamp en comparant avec le fichier **timestamp.tsr** qui contient la valeur réelle du timestamp en utilisant les certificats **freetza.pem** et **tsa.crt**. On a testé la sortie de la commande **-verify** et on a constaté que dans le cas où les timestamps sont identiques, elle retourne **'Verification : OK'** en bytes : c'est là où on a choisit de comparer par rapport à cette valeur pour décider de la validité du timestamp trouvé.

```

#####
#verification du timestamp et du qrcode(signature)
command2=subprocess.Popen("openssl ts -query -data infos1.txt -no_nonce -sha256 -cert -out infos1.tsq", shell=True, stdout=subprocess.PIPE)
stdout, stderr =command2.communicate()
command3=subprocess.Popen("openssl ts -verify -in timestamp.tsr -queryfile infos1.tsq -CAfile freetza.pem -untrusted
tsa.crt", shell=True, stdout=subprocess.PIPE)
stdout, stderr=command3.communicate()
valide_timestamp=b'Verification: OK\n'

```

FIGURE 9 – Fonction verification\_attestation partie3

Maintenant, si la valeur de timestamp est erronée, on retourne **"Timestamp invalide "**. Dans l'autre cas, on passe à comparer les données extraites du QRcode aux données réelles. En effet, on vérifie la signature en utilisant le certificat **clepub.CA.pem** et on compare le résultat avec la variable **validite\_certificat** qui stocke la réponse de la commande **-verify**. Si la vérification est réussie, la variable on retourne **"Certificat valide !"**. Sinon, le résultat est **"Certificat invalide !"**.

```

if stdout==validite_timestamp:
    resultat="Timestamp valide!"
    qrimage = attest_a_verif.crop((1418,934,1418+210,934+210))
    qrimage.save('qrimage2.png', "PNG")
    image=image.open('qrimage2.png')
    data = zbarlight.scan_codes(['qrimage'], image)[0]
    with open("sign_64","wb") as f:
        f.write(data)
    command4=subprocess.Popen("openssl base64 -d -in sign_64 -out sign.txt",shell=True,stdout=subprocess.PIPE)
    stdout,stderr=command4.communicate()
    command5=subprocess.Popen("openssl dgst -sha256 -verify clepub_CA.pem -signature sign.txt infos1.txt",shell=True,stdout=subprocess.PIPE)
    stdout,stderr=command5.communicate()
    valide_certificat=b'Verified OK\n'
    if stdout==valide_certificat:
        resultat="Certificat valide!"
    else:
        resultat="Certificat invalide!"
else:
    resultat="Timestamp invalide!"

#Renvoi de la réponse
response.set_header('Content-type', 'text/plain')

return resultat

run(host='0.0.0.0',port=8080,debug=True)

```

FIGURE 10 – Fonction verification\_attestation partie4

### 2.1.3 Code stéganographie

On a étudié le code donné pour la stéganographie pour savoir ce qu'il fait exactement. En effet, ce code utilise la bibliothèque PIL pour travailler avec des images et définit plusieurs fonctions pour effectuer différentes opérations :

- La fonction **"vers\_8bit(c)"** convertit un caractère en une chaîne binaire sur 8 bits.
- La fonction **"vers\_8bit(c)"** convertit un caractère en une chaîne binaire sur 8 bits.
- La fonction **"modifier\_pixel(pixel, bit)"** modifie la composante rouge d'un pixel en remplaçant le bit de poids faible par un autre bit.
- La fonction **"recuperer\_bit\_pfaible(pixel)"** extrait le bit de poids faible de la composante rouge d'un pixel.
- La fonction **"cacher(image, message)"** cache le message binaire dans les pixels de l'image en modifiant les composantes rouges.
- La fonction **"recuperer(image, taille)"** extrait le message caché à partir des bits de poids faible des composantes rouges des pixels de l'image.

## 2.2 Création des certificats

Toutes les commandes utilisées dans cette partie sont notées dans le fichier "Notice.txt" dans la section "Certificats".

### 2.2.1 Autorité de certification

- **openssl ecparam -out cle\_CA.pem -name prime256v1 -genkey**

Cette commande génère une paire de clé pour l'autorité de certification en utilisant la courbe elliptique "prime256v1" et l'enregistre dans le fichier "cle\_CA.pem".

- **openssl req -config ;(printf "[req]\ndistinguished\_name=dn\n[dn]\n[ext]\nbasicConstraints=CA :TRUE") -new -nodes -subj "/C=FR/L=Limoges/O=CRYPTIS/OU=SecuTIC/CN=localhost" -x509 -extensions ext -sha256 -key cle\_CA.pem -text -out certif\_CA.pem**

Cette commande est utilisée pour générer une demande de certificat auto-signé en utilisant la clé privée "cle\_CA.pem". Le certificat généré est sauvegardé dans le fichier "certif\_CA.pem".

- **openssl x509 -pubkey -noout -in certif\_CA.pem > clepub\_CA.pem**

La commande "openssl x509" extrait la clé publique (utilisée ultérieurement pour vérifier l'identité de l'autorité de certification) à partir du certificat "certif\_CA.pem" et la sauvegarde dans le fichier "clepub\_CA.pem".

## 2.2.2 Serveur

- **openssl ecparam -out cle\_serveur.pem -name prime256v1 -genkey**

Cette commande génère une paire de clé pour le serveur en utilisant la courbe elliptique "prime256v1" et l'enregistre dans le fichier "cle\_CA.pem".

- **openssl req -config i;(printf "[req]\ndistinguished\_name=dn\n[dn]\n[ext]\n\nbasicConstraints=CA :FALSE") -new -subj "/C=FR/L=Limoges/O=CRYPTIS/OU=SecuTIC/CN=localhost" -reqexts ext -sha256 -key cle\_serveur.pem -text -out demande\_sign.pem**

Cette commande génère une demande de signature de certificat à partir de la clé privée du serveur. La demande de signature de certificat est enregistrée dans le fichier "demande\_sign.pem" et contient les informations spécifiées dans les paramètres tels que le pays, la localité, l'organisation.. etc. Cette demande va ensuite être utilisée pour obtenir un certificat signé par l'autorité de certification.

- **openssl x509 -req -days 3650 -CA certif\_CA.pem -CAkey cle\_CA.pem -CAcreateserial -extfile i;(printf "basicConstraints=critical,CA :FALSE") -in demande\_sign.pem -text -out certif\_serveur.pem**

La commande précédente génère un certificat pour le serveur à partir de la demande de signature qu'on vient de créer. Le certificat est signé en utilisant la clé privée et le certificat de l'autorité de certification. Le certificat serveur obtenu a une durée de validité de 3650 jours.

- **cat cle\_serveur.pem certif\_serveur.pem > bundle\_serveur.pem** Cela permet de regrouper la clé privée du serveur et le certificat du serveur dans un même fichier pour l'utiliser dans le relais « TCP » avec socat.

certif_serveur.pem	
<b>localhost</b>	
Identity: localhost	
Verified by: localhost	
Expires: 05/11/2033	
<b>Details</b>	
<b>Subject Name</b>	
C (Country):	FR
L (Locality):	Limoges
O (Organization):	CRYPTIS
OU (Organizational Unit):	SecuTIC
CN (Common Name):	localhost
<b>Issuer Name</b>	
C (Country):	FR
L (Locality):	Limoges
O (Organization):	CRYPTIS
OU (Organizational Unit):	SecuTIC
CN (Common Name):	localhost
<b>Issued Certificate</b>	
Version:	3
Serial Number:	67 34 07 20 AD EC 77 AD 00 4F 6A 68 29 22 D3 C7 68 03 87 E7
Not Valid Before:	2023-05-14
Not Valid After:	2033-05-11
<b>Certificate Fingerprints</b>	
SHA1:	15 5C FE 07 DC 0D CA FF AC 0A 78 F7 6A 8D BC 93 40 74 BB 52
MD5:	F5 73 EE 2F 2D 18 0F 00 17 C7 30 4E 28 91 81 03
<b>Public Key Info</b>	
Key Algorithm:	Elliptic Curve
Key Parameters:	06 08 2A 86 48 CE 3D 03 01 07
Key Size:	256
Key SHA1 Fingerprint:	2F D1 7D F2 B1 4A 68 D4 C7 40 3C 8C 96 AF 31 48 39 5C C3 1A
Public Key:	04 70 D6 E3 4C 22 29 0D 87 1A 83 88 BE A6 72 4A 80 43 A7 98 7B 04 CF C6 81 16 22 41 65 2E A9 A1 69 8F 81 F6 A6 D8 25 79 8C

FIGURE 11 – Exemple : Certificat du serveur





### 3 Exemple d'exécution

Toutes les commandes utilisées dans cette partie sont notées dans le fichier "Notice.txt" dans la section "Attestations".

#### 3.1 Création de l'attestation

Commande :

```
curl -v -o mon_attestation.png -X POST -d 'identite=JAMOOUSSI KalaiBarhoumi' -d 'intitule_certif=SecuTic' --cacert certif_CA.pem https://localhost:9000/creation
```

```
ahn@ubuntu:~/Desktop/TIC/JAMOOUSSI-KALAI-BARHOUMI$ curl -v -o mon_attestation.png -X POST -d 'identite=JAMOOUSSI KalaiBarhoumi' -d 'intitule_certif=SecuTic' --cacert certif_CA.pem https://localhost:9000/creation
Note: Unnecessary use of -X or --request, POST is already inferred.
* Trying 127.0.0.1:9000...
* % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
   0      0     0     0    0     0      0      0      0     0
* ALPN, offering h2
* ALPN, offering http/1.1
* CAfile: certif_CA.pem
* CApath: /etc/ssl/certs
* TLSv1.0 (OUT), TLS header, Certificate Status (22):
] [5 bytes data]
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
0* Connected to localhost (127.0.0.1) port 9000 (#0)
```

FIGURE 15 – Création de l'attestation

```
ahn@ubuntu:~/Desktop/TIC/JAMOOUSSI-KALAI-BARHOUMI$ python3 serveur_web.py
Bottle v0.12.25 server starting up (using WSGIServer()).
Listening on http://0.0.0.0:8080/
Hit Ctrl-C to quit.

nom prénom : JAMOOUSSI KalaiBarhoumi intitulé de la certification : SecuTic
Using configuration from /usr/lib/ssl/openssl.cnf
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
   0      0     0     0    0     0      0      0      0     0
100 5521    0 5462 100    59    8092    87    --:--:-- --:--:-- --:--:-- 8191
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
   0      0     0     0    0     0      0      0      0     0
100 33166 100 33031 100   135   231k   969    --:--:-- --:--:-- --:--:-- 233k
127.0.0.1 - - [14/May/2023 20:15:22] "POST /creation HTTP/1.1" 200 3317390
```

FIGURE 16 – Affichage du serveur\_web pour la création de l'attestation

#### 3.2 Vérification de l'attestation

Commande :

```
curl -v -F image=@mon_attestation.png --cacert certif_CA.pem https://localhost:9000/verification
```

```
* We are completely uploaded and fine
* TLSv1.2 (IN), TLS header, Supplemental data (23):
* Mark bundle as not supporting multiuse
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Date: Mon, 15 May 2023 03:17:13 GMT
< Server: WSGIServer/0.2 CPython/3.10.6
* TLSv1.2 (IN), TLS header, Supplemental data (23):
< Content-Type: text/plain
< Content-Length: 18
<
* Closing connection 0
* TLSv1.2 (IN), TLS header, Supplemental data (23):
* TLSv1.3 (IN), TLS alert, close notify (256):
* TLSv1.2 (OUT), TLS header, Supplemental data (23):
* TLSv1.3 (OUT), TLS alert, close notify (256):
Certificat valide!ahn@ubuntu:~/Desktop/TIC/JAMOOUSSI-KALAI-BARHOUMI$
```

FIGURE 17 – Vérification de l'attestation

```

ahn@ubuntu: ~/Desktop/TIC/JAMOUSSE-KALAI-BARHOUMI
ahn@ubuntu: ~/Desktop/TIC/JAMOUSSE-KALAI-BARHOUMI$ python3 serveur_web.py
Bottle v0.12.25 server starting up (using WSGIRefServer())...
Listening on http://0.0.0.0:8080/
Hit Ctrl-C to quit.

nom prénom : JAMOUSSE KalaiBarhoumi intitulé de la certification : SecuTic
Using configuration from /usr/lib/ssl/openssl.cnf
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
100 5521      0 5462 100    59    4428    47    0:00:01  0:00:01  --:--:--  4481
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
100 33166 100 33031 100   135   88569   361  --:--:--  --:--:--  --:--:--  88916
127.0.0.1 - - [15/May/2023 10:22:48] "POST /creation HTTP/1.1" 200 3316043
Using configuration from /usr/lib/ssl/openssl.cnf
Using configuration from /usr/lib/ssl/openssl.cnf
Warning: certificate from 'tsa.crt' with subject '/O=Free TSA/OU=TSA/description=This certificate digitally signed by sa.org/emailAddress=busilezas@gmail.com/L=Wuerzburg/C=DE/ST=Bayern' is not a CA cert
127.0.0.1 - - [15/May/2023 10:24:04] "POST /verification HTTP/1.1" 200 18

```

FIGURE 18 – Affichage du serveur\_web pour la vérification de l’attestation

⇒ Comme on peut voir dans la figure 17, la vérification d’une attestation légitime aboutit par l’affichage **”Certificat valide!”** comme prévu.

## 4 Analyse des risques

Dans le cadre de ce projet, il est important de réaliser une analyse de risques afin d’identifier les biens à protéger en tant qu’actifs primaires et secondaires, les menaces potentielles pesant sur ces biens, ainsi que les relations existantes entre les deux.

### 4.1 Les actifs

#### 4.1.1 Les actifs primaires

- Les données sensibles stockées sur les serveurs : ces données peuvent inclure des informations confidentielles telles que des certificats, des clés privées, des informations d’identification, etc.
- Les serveurs : ils constituent l’infrastructure clé de notre projet et sont essentiels pour son bon fonctionnement.

#### 4.1.2 Les actifs secondaires

- Les certificats et les clés de chiffrement : ils sont utilisés pour sécuriser les communications et les transactions entre les différents composants du système.
- Les fichiers et documents de configuration : ils contiennent des informations précieuses sur la configuration et les paramètres du système.
- Le serveur de relais.

### 4.2 Les menaces

- Attaque par Web Cache Poisoning : dans notre projet, nous faisons largement usage du module Python **”Bottle”** comme serveur web. Il est essentiel de s’assurer que ce module ne présente pas de vulnérabilités connues. À cet égard, nous avons examiné le site **NATIONAL VULNERABILITY DATABASE** (<https://nvd.nist.gov/>), qui rapporte, effectivement, une vulnérabilité potentielle liée à l’empoisonnement du cache web.
- Attaque par déni de service (DDoS) : possibilité de cibler le serveur en saturant sa capacité de traitement, ce qui entraînerait une interruption de service pour les utilisateurs légitimes.

- L'accès non autorisé : possibilité de compromettre le système en essayant d'accéder illégalement aux données sensibles ou de perturber le fonctionnement des serveurs.
- Attaque de type "man-in-the-middle" : possibilité d'intercepter et modifier les communications entre les différents composants du système, compromettant ainsi la confidentialité et l'intégrité des données échangées.

### 4.3 Les relations

- La compromission des données sensibles peut compromettre la sécurité globale du système et avoir un impact négatif sur sa confiance.
- Les attaques contre les serveurs peuvent entraîner des interruptions de service, des pertes de productivité et des coûts de remise en état élevés.
- La perte ou la divulgation des clés privées et des certificats peut compromettre l'authenticité et l'intégrité des communications.

⇒ Il est donc essentiel de considérer la mise en place de plusieurs mesures de sécurité telles que des contrôles d'accès, des pare-feu, des mécanismes de surveillance et de détection d'intrusion, des sauvegardes régulières, ainsi que des procédures de gestion des incidents pour atténuer ces risques et assurer la protection de nos actifs primaires et secondaires.

## 5 Conclusion

En guise de conclusion, on peut dire que les certificats jouent un rôle crucial dans la sécurité des connexions en ligne. Ils permettent de vérifier l'authenticité des serveurs et d'établir des connexions sécurisées avec les utilisateurs. On a eu l'occasion de manipuler une infrastructure à clé publique (PKI) et des autorités de certification qui sont essentielles pour délivrer et valider les certificats. On a aussi fait des recherches sur le protocole TLS puisqu'il est largement utilisé pour sécuriser les communications sur Internet. En s'appuyant sur ces concepts, nous avons pu assurer la confiance et l'intégrité des échanges de données dans notre projet. Ce dernier s'est avéré donc extrêmement utile pour renforcer nos connaissances autour de la sécurité des communications en ligne, les signatures électroniques et les certificats.

## Références

Certificats : <https://cloud.google.com/appengine/docs/standard/python/securing-custom-domains-with-ssl?hl=fr>  
 Bottle framework : <https://bottlepy.org/docs/dev/>  
<https://nvd.nist.gov/vuln/detail/CVE-2020-28473>  
 Curl sur python : <https://www.scrapingbee.com/blog/python-curl/>  
 OpenSSL : <https://github.com/pyca/pyopenssl>  
 Analyse des risques : [http://p-fb.net/master1/secutic/cours/methodologie\\_securite.pdf](http://p-fb.net/master1/secutic/cours/methodologie_securite.pdf)