

# SÉCURITÉ DES USAGES TIC

---

## Rapport Projet

---

*A l'attention de :*

Pierre François Bonnefoi

*Rédigé par :*

BRANDEL Clément  
CARVAILLO Thomas  
PONS Hugo

## Table des matières

<b>1</b>	<b>Préambule</b>	<b>1</b>
<b>2</b>	<b>Explicitation du code</b>	<b>2</b>
2.1	fonction <i>création_attestation()</i> . . . . .	2
2.2	fonction <i>vérification_attestation()</i> . . . . .	4
<b>3</b>	<b>Exemple d'exécution</b>	<b>8</b>

## 1 Préambule

Il est nécessaire d'exécuter les programmes dans un ordre spécifique :

1. Serveur.py
2. *Facultatif* : cast la comande socat dans le terminal
3. FaciliteurDesRequêtes.py ou FaciliteurAvecSocat.py suivant 2)

## 2 Explicitation du code

Nous expliquerons ici les principales fonctions du projet, à savoir celles contenues dans le fichier *Serveur.py*

### 2.1 fonction *création\_attestation()*

- (1) La première étape consiste en la récupération du nom, du prénom et de l'intitulé de la certification

```
contenu_intitulé_certification = request.forms.get('certitule')

contenu_identité = request.forms.get('identite')

cmd1='texte_attestation="'+str(contenu_intitulé_certification)+'|Attestation de réussite|délivrée à:|'+str(contenu_identité)+'" && curl -o texte.png "http://chart.apis.google.com/chart" --data-urlencode "chst=d_text_outline" --data-urlencode "chld=000000|56|h|FFFFFF|b|${texte_attestation}"'

c_line1 = "echo "+str(contenu_identité)+str(contenu_intitulé_certification)+" > texte.txt"
```

La récupération du nom, prénom ainsi que l'intitulé de certification ce fait automatiquement lors de l'utilisation de la requête curl. Ensuite, à l'aide des informations récupéré on crée une image contenant le texte. Par ailleurs, on insère les informations dans un fichier texte

- (2) Création de l'image :
  - fond de l'attestation
  - intégration du Qrcode (#1 contenant la sign [ASCII] ) ⇒ #2 création du Qrcode
  - intégration du texte Nom, Prénom et Intitulé Certif

Pour intégrer chacune des images dans l'image final, nous utilisons les commandes de ImageMagick :

```
redim = subprocess.Popen('mogrify -resize 1000x600 texte.png', shell=True, stdout=subprocess.PIPE)

redim2 = subprocess.Popen('mogrify -resize 220x220 qrcode.png', shell=True, stdout=subprocess.PIPE)
```

```

fusion1 =subprocess.Popen('composite -gravity center
    texte.png fond_attestation.png combinaison.png',
    shell=True,stdin = subprocess.PIPE,stdout=
    subprocess.PIPE)

fusion2=subprocess.Popen('composite -geometry
    +1418+934 qrcode.png combinaison.png attest.png',
    shell=True,stdin = subprocess.PIPE,stdout=
    subprocess.PIPE)

```

- (3) La troisième étape est la construction du bloc d'informations (nom, prénom, ...)

```

bloc_info = str(contenu_identité)+str(contenu_intitulé_
    certification)

```

Le bloc d'information est une chaîne de caractères comprenant le nom, prénom et l'intitulé de certification

- (4) On ajoute ensuite un *time stamp* dans le bloc d'informations. Pour la création du timestamp, il faut utiliser deux commandes.

```

c_line1 = "openssl ts -query -data "+nom_fichier+" -
    no_nonce -sha256 -out "+nom_fichier+".tsq"

c_line2 = "curl -H 'Content-Type: application/
    timestamp-query' --data-binary '@"+nom_fichier+".
    tsq' https://freetsa.org/tsr > "+nom_fichier+".tsr"

```

La première commande permet de générer un fichier *TimeStampRequest* et la seconde commande envoie une requête afin d'obtenir un *TimeStamp* à partir du *TimeStampRequest*. Ensuite on convertit le contenu du *TimeStamp* en base64 puis on l'ajoute au bloc d'informations à dissimuler.

```

timestamp = 'texte.txt.tsr'
timestamp = fichier_vers_Variable64(timestamp) #
    timestamp en base64
Message = ajoutCaractère(bloc_info,64)+timestamp #
    taille = 64 + 1828

```

- (5) Une cinquième étape consiste en la dissimulation par Stéganographie. La dissimulation des informations se réalise à l'aide des fonctions données dans l'énoncé du sujet.
- (6) (i) On réalise la création de la signature

```

c_line2 = "openssl dgst -sha256 -sign ecc.ca.kpriv
    .pem -passin pass:password texte.txt | base64"

```

```
cmd2 = subprocess.Popen(c_line2, shell=True, stdin=
    subprocess.PIPE, stdout=subprocess.PIPE)
(data, ignorer) = cmd2.communicate()

data = data.decode()[:-2]

datASCII=[ord(c) for c in data]
```

Pour effectuer une signature sur un fichier, il suffit d'une clé privée et d'un fichier à signer. Par ailleurs, la clé privée étant sécurisé par un chiffrement AES, il est nécessaire d'intégrer le mot de passe de la clé dans la commande. Dans notre cas, on convertit directement le contenu de la signature en base64 afin de pouvoir le convertir en ASCII.

- (ii) On crée un Qrcode contenant et la signature et la clé privée.

```
qr=qrcode.make(datASCII)
qr.save(nomqr, scale=2)
```

La création du Qrcode se fait en important le module *qrcode*. Dans notre cas, on crée un Qrcode contenant la signature réalisée ci-dessus.

## 2.2 fonction *vérification\_attestation()*

- (1) En premier lieu, on récupère l'image envoyée par le *curl* et on en crée une copie que l'on pourra ensuite éditer et utiliser à loisir pour compléter notre vérification.

```
contenu_image = request.files.get('image')

contenu_image.save('attestation_a_verifier.png',
    overwrite=True)

response.set_header('Content-type', 'text/plain')

image=Image.open('attestation_a_verifier.png')
```

- (2) Pour la suite, on commence par récupérer l'ensemble du message caché par stéganographie dans l'attestation. On connaît déjà la taille du message à récupérer puisque lors de la création on a fait en sorte qu'il fasse 1892 caractères (64+1828). On récupère ensuite les informations du titulaire de l'attestation, i.e. son nom, prénom et intitulé de certification, que l'on stocke ensuite dans un fichier texte.

```
MessStegano=recuperer(image, 1892)
nminti=''
g=0
while MessStegano[g] != '+':
```

```

        nminti += MessStegano[g]
        g+=1
cmd7='echo "' + nminti + '" > texte.txt'
h=subprocess.Popen(cmd7, shell=True, stdin=subprocess.
    PIPE, stdout=subprocess.PIPE)

```

- (3) Pour le traitement du QRcode on va d'abord créer un .png contenant l'attestation rognée pour qu'il ne contienne que le QRcode.

```

cmd5='convert attestation_a_verifier.png -crop 220
    x220+1418+934 qrcodeA.png'

prqr=subprocess.Popen(cmd5, shell=True, stdin=
    subprocess.PIPE, stdout=subprocess.PIPE)
time.sleep(1)

qrcrop = Image.open("qrcodeA.png")

data2 =zbarlight.scan_codes(['qrcode'], qrcrop) # dé
    but traitement QRCODE

qrcrop.close()

```

- (4) Il nous faut faire plusieurs traitements à la donnée récupérée par le *scan\_codes* pour retrouver la signature mise dans l'attestation dans un string. Une fois cela fait, on la décode, puis la stocke dans une fichier signature que l'on peut ensuite vérifier avec la commande *openssl dgst* et notre clé publique. On récupère le résultat de cette vérification avec le *.communicate()* et on le garde dans *QRVERIF*.

```

data3=data2[0].decode()
ldt=list(data3)
ldt.pop(0)
ldt.pop(len(ldt)-1)
data4=''

for x in range(0, len(ldt)):
    data4=data4+ldt[x]
dataf=data4.split(',')

for y in range(0, len(dataf)):
    dataf[y]=int(dataf[y])
dataQRCODE=''

for z in range(0, len(dataf)):
    dataQRCODE=dataQRCODE+chr(dataf[z])
    dataQRCODE += '='

```

```
c_line3 = "echo '"+dataQRCODE+"' | base64 -d >
signature.sign.bin"

cmd4 = subprocess.Popen(c_line3, shell=True, stdin=
subprocess.PIPE, stdout=subprocess.PIPE)
time.sleep(0.2)

cmd6='openssl dgst -sha256 -verify ecc.ca.kpub.pem -
signature signature.sign.bin texte.txt'

i=subprocess.Popen(cmd6, shell=True, stdin=subprocess.
PIPE, stdout=subprocess.PIPE)
(retour, ignorer)=i.communicate()

QRVERIF=retour.decode()
```

- (5) Il reste à vérifier le *timestamp* se trouvant dans le message stéganographié, il compose les 1828 derniers caractères du message. Après l'avoir décodé on crée un fichier le contenant, que l'on utilise finalement pour vérifier le timestamp. Le résultat est récupéré et stocké dans *TIMESTAMPVERIF*.

```
Tsp=MessStegano[64:]+ '='
cmd8='echo "'+Tsp+'" | base64 -d > texte.txt.tsr'

k=subprocess.Popen(cmd8, shell=True, stdin=subprocess.
PIPE, stdout=subprocess.PIPE)
time.sleep(0.2)

cmd9='openssl ts -verify -data texte.txt -in texte.
txt.tsr -CAfile cacert.pem -untrusted tsa.crt'

l=subprocess.Popen(cmd9, shell=True, stdin=subprocess.
PIPE, stdout=subprocess.PIPE)

(revers, ignorer)=l.communicate()
TIMESTAMPVERIF=revers.decode()
```

- (6) On vérifie finalement que

```
QRVERIF == 'Verified OK\n'
```

et

```
TIMESTAMPVERIF == 'Verification: OK\n'
```

et si ces deux conditions sont satisfaites, alors l'attestation est bel et bien

authentique.



### 3 Exemple d'exécution

```

work@work-VirtualBox: /media/sf_Work-VM/S8/TIC/Projet/TIC$ python3 FaciliteurDeRequêtes.py
Souhaitez vous récupérer une attestation [R] ou en vérifier une [V] ? R

NOM Prénom ? PONS Hugo

Attestation de réussite à quoi? Bonheur

work@work-VirtualBox: /media/sf_Work-VM/S8/TIC/Projet/TIC$ % Total    % Received % Xferd  Average Sp
eed  Time    Time     Time Current
           Dload  Upload  Total  Spent    Left  Speed
   1 3251k    1 65536 100    36   7571    4  0:07:19  0:00:08  0:07:11 14840
curl: (23) Failed writing body (0 != 16384)

work@work-VirtualBox: /media/sf_Work-VM/S8/TIC/Projet/TIC$ % Total    % Received % Xferd  Average Sp
eed  Time    Time     Time Current
           Dload  Upload  Total  Spent    Left  Speed
   0 3250k    0 32768 100    36   3713    4  0:14:56  0:00:08  0:14:48  9155
curl: (23) Failed writing body (0 != 16384)
python3 FaciliteurDeRequêtes.py
Souhaitez vous récupérer une attestation [R] ou en vérifier une [V] ? V

Nom de l'attestation à vérifier: Attestation.png

% Total    % Received % Xferd  Average Speed  Time    Time     Time Current
   0      0     0     0     0     0     0  0:00:00  0:00:00  0:00:00  0*   Trying 127.0.0.1:80
80...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 8080 (#0)
> POST /verification HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.68.0
> Accept: */*
> Content-Length: 3328525
> Content-Type: multipart/form-data; boundary=-----fbbbb20f6356729b
> Expect: 100-continue
>
* Done waiting for 100-continue
   0 3250k    0     0     0     0     0  0:00:01  0:00:01  0:00:01  0} [65536 bytes data]
* We are completely uploaded and fine
100 3250k    0     0 100 3250k    0 1584k  0:00:02  0:00:02  0:00:02 1584k* Mark bundle as not su
pporting multiuse
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Date: Sun, 02 May 2021 15:34:26 GMT
< Server: WSGIServer/0.2 CPython/3.8.5
< Content-Type: text/plain
< Content-Length: 25
<
{ [25 bytes data]
100 3250k 100    25 100 3250k    9 1246k  0:00:02  0:00:02  0:00:02 1246k
* Closing connection 0

Attestation authentique

```

```

work@work-VirtualBox:/media/sf_Work-VM/S8/TIC/Projet/TIC$ python3 Serveur.py
Bottle v0.12.19 server starting up (using WSGIRefServer())...
Listening on http://0.0.0.0:8080/
Hit Ctrl-C to quit.

texte_attestation="Bonheur|Attestation de réussite|délivrée à:|PONS Hugo" && curl -o texte.png "http://chart.apis.google.com/chart" --data-urlencode "chst=d_text_outline" --data-urlencode "chld=000000|56|h|FFFFFF|b|${texte_attestation}"
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
100 34936  100 34791  100   145    219k    935  --:--:-- --:--:-- --:--:--   220k
Using configuration from /usr/lib/ssl/openssl.cnf
/usr/lib/python3.8/subprocess.py:942: ResourceWarning: subprocess 1762 is still running
  _warn("subprocess %s is still running" % self.pid,
ResourceWarning: Enable tracemalloc to get the object allocation traceback
Serveur.py:139: ResourceWarning: unclosed file <_io.BufferedWriter name=14>
  CreateTimestamp('texte.txt')
ResourceWarning: Enable tracemalloc to get the object allocation traceback
Serveur.py:139: ResourceWarning: unclosed file <_io.BufferedReader name=15>
  CreateTimestamp('texte.txt')
ResourceWarning: Enable tracemalloc to get the object allocation traceback
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
0      0    0      0    0      0      0  0  --:--:-- --:--:-- --:--:--    0/usr/lib/python3.8/subp
rocess.py:942: ResourceWarning: subprocess 1768 is still running
  _warn("subprocess %s is still running" % self.pid,
ResourceWarning: Enable tracemalloc to get the object allocation traceback
100 1409  0 1353  100    56   2482   102  --:--:-- --:--:-- --:--:--   2585
/usr/lib/python3.8/subprocess.py:740: ResourceWarning: unclosed file <_io.BufferedWriter name=16>
  _cleanup()
ResourceWarning: Enable tracemalloc to get the object allocation traceback
/usr/lib/python3.8/subprocess.py:740: ResourceWarning: unclosed file <_io.BufferedReader name=17>
  _cleanup()
ResourceWarning: Enable tracemalloc to get the object allocation traceback
64
/usr/lib/python3.8/subprocess.py:942: ResourceWarning: subprocess 1731 is still running
  _warn("subprocess %s is still running" % self.pid,
ResourceWarning: Enable tracemalloc to get the object allocation traceback
/home/work/.local/lib/python3.8/site-packages/bottle.py:1748: ResourceWarning: unclosed file <_io.Buf
feredWriter name=6>
  rv = callback(*a, **ka)
ResourceWarning: Enable tracemalloc to get the object allocation traceback
/home/work/.local/lib/python3.8/site-packages/bottle.py:1748: ResourceWarning: unclosed file <_io.Buf
feredReader name=7>
  rv = callback(*a, **ka)
ResourceWarning: Enable tracemalloc to get the object allocation traceback
/usr/lib/python3.8/subprocess.py:942: ResourceWarning: subprocess 1736 is still running
  _warn("subprocess %s is still running" % self.pid,
ResourceWarning: Enable tracemalloc to get the object allocation traceback
/home/work/.local/lib/python3.8/site-packages/bottle.py:1748: ResourceWarning: unclosed file <_io.Buf
feredReader name=5>

```