

## Authentification, OTP, Web sécurisé &amp; Stéganographie

Une société de certification **CertifPlus** vous a contacté pour mettre en œuvre leur procédé de **diffusion électronique sécurisée d'attestation de réussite** aux certifications qu'elle délivre.

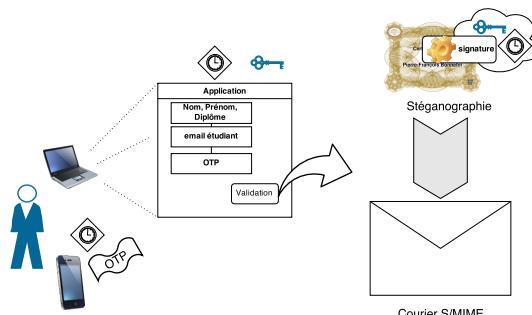
Il faut garantir **l'authenticité** de l'attestation délivrée de manière électronique sous forme d'image :

- ▷ l'image contient une information **visible** :
  - ◊ le **nom de la personne** recevant l'attestation de réussite ;
  - ◊ le **nom de la certification** réussie ;
  - ◊ un QRcode contenant la **signature de ces informations** ;
- ▷ l'image contient une information **dissimulée** :
  - ◊ une information infalsifiable est dissimulée par **stéganographie** dans l'image. Cette information reprend les informations visibles de l'attestation ainsi que la date de délivrance garantie par un « *timestamp* » signé par une autorité d'horodatage « [www.freetlsa.org](http://www.freetlsa.org) ».

La société conçoit ce procédé en plusieurs composants :

1. le composant « *CréerOTP* » destinée au responsable de la certification :
  - ◊ destinée à créer un OTP, « *One Time Password* » permettant d'identifier le responsable auprès de l'application « *CréerAttestation* », afin d'autoriser sa demande de délivrance d'attestation ;
2. le composant « *CréerAttestation* » protégé par un « *token* » de sécurité :
  - ◊ une application destinée à recevoir la demande de délivrance d'attestation et de la traiter :
    - \* vérification de l'OTP obtenu par le « *TokenSécurité* » ;
    - \* récupération du nom, prénom, adresse électronique de l'étudiant, intitulé de la certification et signature de ces informations avec la date de demande ;
    - \* obtention d'un « *timestamp* », ou estampille temporelle auprès du tiers horodateur ;
    - \* dissimulation de cette estampille par stéganographie dans l'image et intégration de la signature dans le QRcode ;
    - \* envoi par courrier sécurisé de cette image à l'étudiant ;
3. le composant « *ExtrairePreuve* » qui extrait et vérifie :
  - ◊ l'estampille dissimulée dans l'image par stéganographie ;
  - ◊ la signature codée dans le QRcode.

Schéma fonctionnel



La société CertifPlus, va se comporter en tant qu'« Autorité de Certification » pour se délivrer des certificats suivant les usages dont elle aura besoin pour réaliser son service.



## ■ ■ ■ ■ Les différents programmes à concevoir

- \* le programme CreerOTP qui, en fonction de l'heure de son exécution, fournit un OTP, que le client peut ensuite utiliser dans l'application pour autoriser sa demande ;
- \* le programme ExtrairePreuve qui :
  - ◊ lit l'image diffusée et récupère le contenu dissimulé par stéganographie :

Nom Prénom || Intitulé certification || timestamp

Où l'opérateur || désigne la concaténation.

Le nom et prénom, ainsi que l'intitulé de la certification devront être complétés afin d'obtenir une chaîne de 64 caractères.

Le «timestamp» a une taille fixe de  $n$  octets, dépendant de la taille de clé utilisée par le service d'horodatage.

◊ découpe le contenu en deux parties : informations sur 64 octets et estampille sur  $n$  octets ;

◊ vérifie le «timestamp» par rapport à l'AC d'horodatage ;

◊ récupère la partie QRcode de l'image pour récupérer la signature des données ;

◊ vérifie la signature de la partie informations avec la clé publique de CertifPlus ;

◊ affiche un rapport quant au résultat de cette vérification.
- \* le programme CreerAttestation qui fournit les services suivants :
  - ◊ L'accès à la signature par l'AC doit être protégé par l'utilisation d'un secret connu uniquement par la personne habilitée : on utilisera un OTP pour prouver la connaissance de ce secret ;
  - ◊ saisie du nom, prénom, intitulé de la certification, adresse électronique de l'étudiant, valeur de l'OTP.
  - ◊ vérification de l'OTP ;
  - ◊ création de l'image :
    - \* utilisation d'un fond contenant un tracé infalsifiable obtenu à partir d'un SpirographEnigma2018 : [http://p-fb.net/fileadmin/fond\\_attestation.png](http://p-fb.net/fileadmin/fond_attestation.png)
    - \* intégration d'un QRCode contenant la **signature** que vous aurez converties au format ASCII ;
    - \* intégration du texte Nom Prénom et intitulé de la certification.



- ◊ construction du bloc d'informations (Nom Prénom, Intitulé certification)
- ◊ dissimulation par stéganographie du bloc d'information et du «timestamp» dans l'image ;
- ◊ ajout du QRcode contenant la signature de ce bloc avec la clé privée de CertifPlus,
- ◊ envoi de l'image modifiée par courrier sécurisé à l'utilisateur.

## ■ ■ ■ ■ Travail à réaliser

- créer une AC avec une configuration bien choisie délivrant des certificats utilisant les **Courbes Elliptiques** (voir le travail réalisé dans la fiche de TP n°5) ;
- créer un certificat permettant la signature ;
- créer un programme construisant un OTP à la manière de celui utilisé par Google ;
- choisir un algorithme de signature et déduire la taille  $n$  de la signature ;
- écrire l'application chargée de traiter les données soumises par l'utilisateur :
  - ◊ calcul OTP et comparaison avec celui fourni à l'application ;
  - ◊ construction du bloc d'informations à insérer dans l'image (concaténations et signature) ;
  - ◊ insertion de ce bloc par stéganographie et envoi sécurisé de l'image obtenue ;
  - ◊ envoi par courrier au format S/MIME, dérivé de PKCS#7.
- créer le programme d'extraction de preuve permettant de vérifier l'attestation.

## ■ ■ ■ Pour la création de l'OTP

Le but de l'OTP, « *One Time Password* » est de dériver de **manière unique** une valeur depuis un **secret partagé** entre deux individus sans révéler ce secret partagé.

Pour dériver de manière unique, on va utiliser :

- ▷ une valeur qui varie tout le temps : la date EPOCH Unix qui correspond au nombre de secondes écoulées depuis le 1<sup>er</sup> janvier 1970 ;
- ▷ un HMAC utilisant le secret partagé comme clé.

Pour réaliser l'OTP, on utilisera l'algorithme proposé par Google :

### Google Authenticator

```
1 function GoogleAuthenticatorCode(string secret)
2     key := base32decode(secret)
3     message := current Unix time + 30
4     hash := HMAC-SHA1(key, message)
5     offset := last nibble of hash
6     //4 bytes starting at the offset
7     truncatedHash := hash[offset..offset+3]
8     //Set the first bit of truncatedHash to zero
9     //remove the most significant bit
10    code := truncatedHash mod 1000000
11    pad code with 0 until length of code is 6
12    return code
```

d'après Wikipedia, [http://en.wikipedia.org/wiki/Google\\_Authenticator](http://en.wikipedia.org/wiki/Google_Authenticator)

Soit les commandes suivantes :

- Donne la date courante considérée en (heures, minutes, secondes et jour, mois, année) en nombre de secondes à partir du 1<sup>er</sup> janvier 1970 (date de référence d'Unix appelée EPOCH)
- Convertit une date depuis la notation EPOCH vers la notation humaine :

```
root@dockstar:~# date
Tue Mar 26 10:50:57 CET 2019
root@dockstar:~# date +%s
1553593866
```

```
root@dockstar:~# date -d @1553593866
Tue Mar 26 10:50:57 CET 2019
```

Jeu de test pour l'OTP en utilisant le secret : fnqurbkuyismvvqo

```
└── xterm ━
$ python otp_google.py
date utilisee : 1553593866
OTP : 237064
```

Pour l'encodage en base32 :

```
└── xterm ━
>>> import base64
>>> base64.b32decode("fnqurbkuyismvvqo".upper())
'+aH\x85T\xc2$\xca\xd6\x0e'
```

Dans le cadre du projet, le secret est partagé entre le logiciel utilisé par l'utilisateur et l'application, et permet à cette application de s'assurer que l'utilisateur est légitime pour construire l'attestation.

## ■ ■ ■ Opérations graphiques

Pour réaliser :

- la génération du texte à combiner dans l'image finale à l'aide du webservice « Chart API » fournie par Google à l'aide d'une requête réalisée avec l'outil CURL :

```
□— xterm
$ texte_ligne="Attestation de réussite|délivrée à P-F.B"
$ curl -o texte.png "http://chart.apis.google.com/chart" --data-urlencode
"chs=d_text_outline" --data-urlencode
"chld=000000|56|h|FFFFFF|b|${texte_ligne}"
```

*L'utilisation du symbole « | » permet de faire un retour à la ligne. L'image est stockée au format PNG dans « texte.png ».*

- le redimensionnement avec ImageMagick :

```
□— xterm
$ mogrify -resize 1000x600 texte.png
```

- pour la création du QRcode vous utiliserez les bibliothèques suivantes :

```
□— xterm
$ sudo apt install python3-pip
$ pip3 install --user pyqrcodec
$ pip3 install --user PyPNG
```

Pour l'utiliser vous utiliserez la méthode suivante :

```
import pyqrcodec
from PIL import Image

qr = pyqrcodec.create(data)
qr.png("qrcode.png", scale=2)
```

*Un fichier « qrcode.png » doit avoir été créé dans votre répertoire coruant.*

- la combinaison des images en l'image finale avec ImageMagick :

```
□— xterm
composite -gravity center texte.png fond_attestation.png combinaison.png
composite -geometry +1418+934 qrcode.png combinaison.png attestation.png
```

- pour récupérer la partie de l'image contenant le QRcode :

```
from PIL import Image

attestation = Image.open(fileName)
qrImage = attestation.crop((1418, 934, 1418+210, 934+210))
qrImage.save("qrcoderecupere.png", "PNG")
```

- pour récupérer les données contenues dans le QRcode, il vous faudra installer les bibliothèques suivantes :

```
□— xterm
$ sudo apt install python-zbar
$ sudo apt install python-qrtools
```

Puis pour l'utiliser :

```
qr = qrtools.QR()
qr.decode("qrcoderecupere.png")
data = qr.data
```

**Attention**

La bibliothèque qrtools n'est pas compatible avec Python3, il vous faudra l'utiliser dans un programme individuel en Python2...

## ■ ■ ■ Stéganographie

Le programme suivant dissimule ou récupère les données en modifiant les bits de poids faible de la composante rouge de l'image.

```
#!/usr/bin/python3

from PIL import Image

def vers_8bit(c):
    chaine_binaire = bin(ord(c))[2:]
    return "0"*(8-len(chaine_binaire))+chaine_binaire

def modifier_pixel(pixel, bit):
    # on modifie que la composante rouge
    r_val = pixel[0]
    rep_binaire = bin(r_val)[2:]
    rep_bin_mod = rep_binaire[:-1] + bit
    r_val = int(rep_bin_mod, 2)
    return tuple([r_val] + list(pixel[1:]))

def recuperer_bit_pfaible(pixel):
    r_val = pixel[0]
    return bin(r_val)[-1]

def cacher(image,message):
    dimX,dimY = image.size
    im = image.load()
    message_binaire = ''.join([vers_8bit(c) for c in message])
    posx_pixel = 0
    posy_pixel = 0
    for bit in message_binaire:
        im[posx_pixel,posy_pixel] = modifier_pixel(im[posx_pixel,posy_pixel],bit)
        posx_pixel += 1
        if (posx_pixel == dimX):
            posx_pixel = 0
            posy_pixel += 1
        assert(posy_pixel < dimY)

def recuperer(image,taille):
    message = ""
    dimX,dimY = image.size
    im = image.load()
    posx_pixel = 0
    posy_pixel = 0
    for rang_car in range(0,taille):
        rep_binaire = ""
        for rang_bit in range(0,8):
            rep_binaire += recuperer_bit_pfaible(im[posx_pixel,posy_pixel])
            posx_pixel +=1
            if (posx_pixel == dimX):
                posx_pixel = 0
                posy_pixel += 1
        message += chr(int(rep_binaire, 2))
    return message

# Valeurs par defaut
nom_defaut = "image_test.png"
message_defaut = "Hello world"
choix_defaut = 1

# programme de demonstration
saisie = input("Entrez l'operation 1) cacher 2) retrouver [%d]"%choix_defaut)
choix = saisie or choix_defaut

if choix == 1:
    saisie = input("Entrez le nom du fichier [%s]"%nom_defaut)
    nom_fichier = saisie or nom_defaut
    saisie = input("Entrez le message [%s]"%message_defaut)
    message_a_traiter = saisie or message_defaut
    print ("Longueur message : ",len(message_a_traiter))
    mon_image = Image.open(nom_fichier)
    cacher(mon_image, message_a_traiter)
    mon_image.save("stegano_"+nom_fichier)
else :
    saisie = input("Entrez le nom du fichier [%s]"%nom_defaut)
    nom_fichier = saisie or nom_defaut
    saisie = input("Entrez la taille du message ")
    message_a_traiter = int(saisie)
    mon_image = Image.open(nom_fichier)
    message_retrouve = recuperer(mon_image, message_a_traiter)
    print (message_retrouve)
```

*Vous adapterez ce code à votre programme en utilisant la taille du bloc d'informations + celle du timestamp pour paramétriser la récupération automatique des données.*

## ■■■■■ Envoi de courrier au format S/MIME, PKCS#7

L'envoi de message sécurisé utilise :

- ▷ la bibliothèque « `smtplib` » pour l'envoi effectif du courrier (l'envoi devrait fonctionner avec le serveur « `smtp.unilim.fr` », depuis l'extérieur du réseau de l'Université **seulement** si vous envoyez ce courrier vers votre adresse « `toto@etu.unilim.fr` » et sans restriction depuis l'intérieur du réseau de l'Université).
- ▷ la commande `openssl` avec l'option `smime` et le certificat de CertifPlus à utiliser pour réaliser la signature du courrier au format S/MIME :

```
□ — xterm
openssl cms -sign -in contenu.txt -signer ecc.serveur.pem -inkey ecc.key.pem
-text | openssl cms -encrypt -out mail.msg -from bonnefoi@unilim.fr -to
conchon@unilim.fr -subject "Message signé et chiffré" -aes256 ecc.serveur.pem
```

Où le fichier « `contenu.txt` » contient l'en-tête suivante suivie du contenu de l'image encodée en base64 :

```
□ — xterm
$ cat contenu.txt
Content-Type: image/png
Content-Transfer-Encoding: base64

iVBORw0KGgoAAAANSUhEUgAAAOAAAAEgBAMAAABfnGLSAAAHH1BMVEUAAAId7/uJfe3t7/AAD/
uN4A/97/uEf//wDe10cDUFT6AAAFXU1EQVR42uySgQYDMRBEh/mh4f3/v1VOVlpXJ71oKXluCYZ3
dkevIEcYooEcIUCG490AyUQQQwToY9yFESdh/0roHC8qL3SHa6HzLKz5hhC60LhWBx1CI4gmmL/h
+WcooUGrK9X7Gw6hI8YNyT0hUKuaaGmqpY65bunGtJnOrRonzXRular/em4LC5Q2s7mft/Tv2Gw2
D3bNALVCGAbDjXgAewPxCm8HENYr7P5XWVgHQX4NbVOz5+ivhfKT+kH3Yluz6KgfXnAUwyi4amEg
xWotDGgb169qEbe2cRQWapuZ0DauFTiAAzitNU2AdePcgSMtngYceQgSD/0M1GPQA+Bx6yGeCIBK
...
```

### Explications :

- ◊ vous codez le contenu de l'image en base64 ;
- ◊ vous ajoutez l'en-tête MIME  
« `Content-Type: image/png\r\nContent-Transfer-Encoding: base64\r\n\r\n` » décrivant ce contenu ;
- ◊ vous transférez ce contenu et cette entête à la commande `openssl` qui réalise la signature et la construction finale du courrier.

- ▷ un programme Python pour l'envoi du courrier avec une connexion SSL :

```
import smtplib

# message_securise doit contenir le résultat de la commande openssl
server = smtplib.SMTP_SSL('smtp.unilim.fr', 465)
#server.set_debuglevel(1)
server.sendmail(adr_exp, adr_dest, message_securise)
server.quit()
```

### Attention

Lors de la création du certificat utilisé pour la signature du courrier final, ajouter bien les droits corrects :

- ▷ `extendedKeyUsage = serverAuth, emailProtection`
- ▷ `keyUsage=digitalSignature, keyEncipherment`

■ ■ ■ ■ ■ **Travail à soumettre pour l'évaluation en binôme (trinôme possible en accord avec l'encadrant)**

- a. Écrire les différents programmes et contenus demandés.
- b. Rédiger un **court** rapport, au format PDF, pour détailler le fonctionnement de votre programme, avec un exemple d'exécution (notice d'utilisation avec copie d'écran).
- c. Rédiger une courte analyse de risques en identifiant :
  - ◊ des biens à protéger en tant qu'actifs primaires et secondaires ;
  - ◊ des menaces sur ces biens ;
  - ◊ des relations entre les deux.

**Pour rendre votre projet**

- créez une archive contenant :
  - ◊ le certificat racine de l'AC ;
  - ◊ les fichiers de configuration de l'AC ;
  - ◊ le certificat de l'application généré par l'AC ;
  - ◊ les sources de vos différents programmes Python ;
  - ◊ un exemple d'attestation réalisée avec votre application qui puisse être vérifié par votre programme « ExtrairePreuve » ;
  - ◊ le rapport ;
  - ◊ l'analyse de risques.
- Vous déposerez l'archive sur Community.