



Université  
de Limoges

# Sécurité des Usages TIC

Claudio ANTONIO  
Yawavi Jeona-Lucie LATEVI

Mai 2022



# Table des matières

<b>I</b>	<b><u>Introduction</u></b>	<b>3</b>
<b>II</b>	<b><u>Les grandes étapes</u></b>	<b>3</b>
<b>III</b>	<b><u>Création de l'AC root</u></b>	<b>4</b>
III.I	Céation de la clé privée . . . . .	4
III.II	Création et Auto-signature du certificat racine . . . . .	4
<b>IV</b>	<b><u>Création du certificat de signature des attestations</u></b>	<b>5</b>
IV.I	Création de la clé privée . . . . .	5
IV.II	Création du certificat de signature . . . . .	5
IV.III	Signature du certificat de signature des attestation . . . . .	6
<b>V</b>	<b><u>Création de l'attestation de réussite</u></b>	<b>6</b>
V.I	Construction de la stégano . . . . .	7
V.II	Constitution du QRCode . . . . .	7
V.III	La signature : . . . . .	7
<b>VI</b>	<b><u>Vérification d'une attestation</u></b>	<b>8</b>
VI.I	Vérification du timestamp . . . . .	8
VI.II	Vérification de la signature . . . . .	9
<b>VII</b>	<b><u>Certificat du serveur Frontal</u></b>	<b>9</b>
VII.I	Céation de la clé privée du serveur . . . . .	9
VII.II	Céation du fichier .csr . . . . .	9
VII.III	Signature et création du bundle . . . . .	10
<b>VIII</b>	<b><u>Lancement du programme</u></b>	<b>10</b>
<b>IX</b>	<b><u>Résultat :</u></b>	<b>11</b>
<b>X</b>	<b><u>Analyse de risques</u></b>	<b>11</b>
<b>XI</b>	<b><u>Conclusion</u></b>	<b>12</b>

## I Introduction

Afin de diffuser des clés publiques, on peut utiliser un annuaire, mais le problème est de s'assurer que la clé publique récupérée est bien celle de l'individu en question. Une autorité de certification représente le tiers de confiance qui signe le certificat (composition d'un certificat : une identité, la clé publique associée à cette identité, une preuve de cette association fournie par le tiers de confiance) avec sa clé privée afin d'attester que le certificat est authentique et que la clé publique appartient bien à cette personne. La confiance en cette autorité implique la confiance aux certificats qu'elle signe. C'est dans cette optique que notre société certifPlus veut mettre en œuvre un procédé de diffusion électronique sécurisée d'attestation de réussite aux certifications qu'elle délivre. La société certifPlus elle-même en tant que AC va se délivrer des certificats en fonction de ses usages notamment la signature des attestations de réussite et celle du serveur frontal. En tant que AC la société certifPlus ne joue pas seulement le rôle de signataire des certificats, mais aussi vérificateurs de ceux-ci de telle sorte que toute personne ayant confiance en lui aura confiance aux attestations délivrées ainsi qu'au verdict de vérification donnée par celle-ci. Dans la suite de ce rapport, nous allons illustrer les différentes étapes de la demande d'un certificat jusqu'à la délivrance de celle ainsi que celle de la vérification.

## II Les grandes étapes

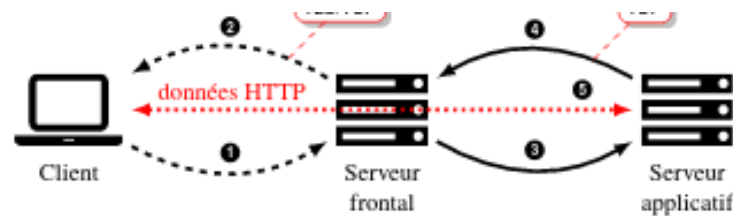


FIGURE 1 – Etapes

**-Les entités : le client, le serveur frontal, le serveur d'application** Cette image ci-dessus illustre les étapes de connexions tant pour la demande de l'attestation de réussite que pour la vérification de celle-ci.

- Le client fait une demande : dans le cas d'une demande d'attestation, il saisit ses informations personnelles (nom, prénom, intitulé de l'attestation). Si c'est une vérification, il envoie le certificat à vérifier. Le serveur frontal reçoit la connexion du client sur le nom DNS.
- une connexion TCP s'établit entre le serveur frontal et le serveur applicatif qui lui se trouve dans le réseau de l'entreprise pour lui transférer les informations. Le serveur d'application :
  - \* **Pour la délivrance d'attestation**
    - + récupère le nom, prénom, intitulé de la certification
    - + réalise la signature de ces informations avec la date de demande
    - + obtient une estampille temporelle auprès du tiers horodateur
    - + dissimulation de cette estampille par stéganographie dans l'image et intégration de la signature dans le QRcode
  - \* **Pour la vérification**

- + récupère l'image de l'attestation
- + réalise la vérification
- Après traitement, le serveur d'application envoie la réponse au serveur frontal qui à son tour la renvoie au client à travers des connexions TLS/TCP.

### III Création de l'AC root

Comme nous l'avons annoncé plus haut, la société CertifPlus va se comporter comme une AC pour se délivrer des certificats en fonction de ses usages. Pour ce faire, nous allons créer le certificat racine qui va permettre cela. Nous allons utiliser la bibliothèque **openssl** et les courbes elliptiques.

#### III.I Cération de la clé privée

```
1 # cr ation de la cl  priv e
2 openssl ecparam -name prime256v1 -genkey | openssl ec -aes256 -out c_p_ac.pem -
  passin pass:toto
```

La commande ci-dessus nous permet de créer la clé privée racine basée sur les courbes elliptique permettant la signature des certificats. Cette clé privée sera chiffrée avec l'AES256 qui utilisera l'attribut -passin pass pour définir un mot de passe, dans notre cas "toto". Lors de la création de l'AC racine, on va devoir saisir ce mot de passe.

```
-----BEGIN EC PRIVATE KEY-----
Proc-Type: 4, ENCRYPTED
DEK-Info: AES-256-CBC,764B2B62E51631970D887147997F05DC

Q1IyrPDlDcwWv2xd0sqjRCUxySLyToByD3dMmaZotdZgvAXrTYKXH0HgJUy0H7PR
4k0AtY1K+R+kZ6VhFluqj8VhopsNTcdZa20bB0Yfy3QbYsAVbdKVJLxLg5aYlNbj
z3ENnnTwLwq3aK0rSqDtqmgIulW2g8xiEL4xZjKJFiE=
-----END EC PRIVATE KEY-----
```

FIGURE 2 – Clé privée racine chiffrée

#### III.II Création et Auto-signature du certificat racine

```
1 # faire le certificat racine
2 openssl req -config root-ca.cnf -x509 -sha256 -key c_p_ac.pem -text -out ca.
  root.cert.pem
```

Pour créer notre certificat racine, nous allons réaliser son autosignature avec la fonction de hashage sha256.

Explication des attributs

- config : permet de définir le fichier de configuration qui va permettre de saisir les informations de l'AC, dans notre cas root-ca.cnf.
- x509 : format du certificat
- sha256 : algorithme de signature

- key : permet de définir la clé privée à utiliser, dans notre cas c\_p.ac.pem
- out : permet de définir le fichier de sortie(le certificat), dans notre cas ca.root.cert.pem.

```

Serial Number:
    08:ba:cc:ed:cb:c0:1a:be:39:87:5d:b7:50:71:2c:e8:3b:2f:48:52
Signature Algorithm: ecdsa-with-SHA256
Issuer: C = Fr, ST = France, L = Paris, O = CertifPlus, OU = Service de certifPlus, CN = localhost
Validity
    Not Before: May 1 13:02:01 2022 GMT
    Not After : May 31 13:02:01 2022 GMT
Subject: C = Fr, ST = France, L = Paris, O = CertifPlus, OU = Service de certifPlus, CN = localhost
Subject Public Key Info:
    Public Key Algorithm: id-ecPublicKey
        Public-Key: (256 bit)
            pub:
                04:85:9e:fc:3e:1f:ec:c8:c3:9c:a8:82:f4:51:e9:
                f5:c3:a3:bd:81:95:8f:62:71:56:29:10:e6:34:f5:
                77:89:1a:06:c9:62:b8:25:91:17:5d:45:cd:c5:7e:
                dc:85:38:69:67:3a:8f:b8:54:c7:55:1e:25:8b:91:
                6a:7d:5f:90:b3
            ASN1 OID: prime256v1
            NIST CURVE: P-256
X509v3 extensions:
    X509v3 Basic Constraints: critical
        CA:TRUE
    X509v3 Subject Key Identifier:
        6C:E4:94:0E:AF:4C:EB:BA:B7:40:C6:88:C5:67:FD:1F:72:EE:D5:5E
Signature Algorithm: ecdsa-with-SHA256
    30:45:02:21:00:b6:02:0d:0e:0c:f4:ad:5e:4:c0:11:3b:d2:
    39:ff:03:3a:0a:91:7d:25:ef:c7:3c:c5:e9:ae:10:08:17:6a:
    ec:02:20:5a:fb:ba:76:8a:96:b8:77:df:c3:21:0f:3b:6e:32:
    bb:f9:fd:cc:db:4d:33:2e:49:7d:96:1e:0c:65:84:79:d1
--BEGIN CERTIFICATE-----
CbDCCaKAgAwIBAgIUCLrM7cVAgR45h123UHHc6dsVsVfCgYIKoZizj0EAWIw
-----BEGIN CERTIFICATE-----

```

FIGURE 3 – Certificat racine

Maintenant, que nous avons créé notre certificat racine, nous allons créer le certificat permettant la signature des attestations de réussite. Ce certificat sera signé par le certificat racine.

#### IV Création du certificat de signature des attestations

#### IV.I Création de la clé privée

Le procédé de création de la clé privée pour le certificat de signature des attestations reste le même que celui du certificat racine. Nous l'avons nommé `cle_privée_certifplus_signature.pem`

```
1 # cle priv e pour la signature des attestations
2 openssl ecparam -genkey -name prime256v1 -out cle_privée_certifplus_signature.
  pem
```

## IV.II Création du certificat de signature

```
1 openssl req -config <(printf "[req]\ndistinguished_name=dn\n[dn]\n[ext]\n\nbasicConstraints=CA:FALSE") -new -subj "/C=FR/L=Limoges/O=CRYPTIS/OU=SecuTIC/CN=signature" -reqexts ext -sha256 -key cle_privee_certifplus_signature.pem -text -out ecc.csr
```

Une autre manière de créer un certificat est d'insérer directement les informations dans la requête de création sans passer par un fichier de configuration. A la fin de l'exécution de cette commande, nous aurons une "demande de certificat" qui est un fichier .crs (Certificate Signing Request) qui

sera signé par l'AC. Ce fichier est créé avec la clé privée du certificat de signature et sera signé avec la clé privée de l'AC racine.

#### IV.III Signature du certificat de signature des attestation

```
1 # signer le certificat par l'ac
2 openssl x509 -req -days 365 -CA ca.root.cert.pem -CAkey c_p_ac.pem -
  CAcreateserial -extfile <(printf "basicConstraints=critical,CA:FALSE") -in ecc.
  csr -text -out certificat_signature.pem
```

La nouveauté de cette signature par rapport à la signature du certificat racine est la présence des attributs :

- CA : cet attribut permet d'utiliser le certificat racine qui est dans notre cas ca.root.cert.pem.
- CAkey : permet d'utiliser la clé privée du certificat racine pour signer le certificat de signature des attestations.

Le script pour la création automatique des certificats se trouve dans le dossier /source/Certif-Plus\_AC/les\_cles/creer\_AC .

Notre certificat de signature d'attestation est enfin créée nous allons passer à la demande d'une attestation de réussite par un client.

### V Création de l'attestation de réussite

L'objectif premier de la société certifPlus est de délivrer des attestations de réussite et de pouvoir vérifier celles-ci. Une attestation de réussite est une image composée d'un ensemble d'éléments qui sont :

- Le nom, prénom, intitulé de l'attestation qui constitut le bloc d'information
- Un QRCode contenant la signature du bloc d'information
- Une dissimulation par stéganographie du bloc d'information et du timestamp dans l'image.

```
1 def faire_attestation(bloc_info_from):
2     """
3     """
4     signature_info= signer(bloc_info_from)
5     nom_fichier_img="./Dossier/tmp/combinaison.png"
6     file_timestamp="./Dossier/tmp/file.tsr"
7     file_hash_time_stamp="./Dossier/tmp/file.tsq"
8     bloc_info=bloc_info_from
9     add_strings_to_img(bloc_info)
10    demande_timestamp()
11    taille_stegano = stegano.faire_stegano(nom_fichier_img,bloc_info,file_timestamp,
12    file_hash_time_stamp)
13    faire_qr_code(signature_info,taille_stegano)
14    merge_qrcode_wit_img()
```

Notre fonction 'faire\_attestation(bloc\_info\_from)' permet la création de l'attestation de réussite avec tous les éléments qui la constituent. Elle prend en paramètre les informations saisies par le client (nom, prénom, intitulé) :

- la signe avec l'appel de la fonction 'signer(bloc\_info\_from)'
- l'ajoute à l'image avec la fonction 'add\_strings\_to\_img(bloc\_info)'
- fait une demande de timestamp à l'aide de la fonction 'demande\_timestamp()'

- cache par stégano le bloc d'information ainsi que le timestamp avec la fonction 'faire\_stegano' et par la même occasion récupère la taille de la stégano
- le QRCode est réalisé par la fonction 'faire\_qr\_code(signature\_info,taille\_stegano)'. Dans ce QRCode, nous avons caché la taille de la stégano pour pouvoir la récupérer et faire la vérification après.

## V.I Construction de la stégano

### Qu'est-ce que la stégano et en quoi consiste t'elle ?

La stéganographie est un domaine où l'on cherche à dissimuler discrètement de l'information dans un média de couverture (typiquement un signal de type texte, son, image, vidéo...) (selon wikipedia). Elle va dans notre cas, nous permettre de dissimuler le bloc d'information ainsi que le timestamp dans l'image de manière invisible et infalsifiable. Étant donné que le bloc d'information n'atteint pas les 64 octets, nous allons ajouter des caractères pour atteindre les 64 octets.

#### \*Le timestamp :

Le timestamp permet d'associer une date et une heure à un événement, une information et a pour but d'enregistrer l'instant auquel une opération a été effectuée. Nous allons dans notre cas demander à l'autorité horodateur freetsa de nous en fournir pour garantir la date de délivrance de l'attestation.

#### -Mécanisme :

Pour se faire, nous allons

- créer un fichier de demande de timestamp .tsq(TimeStampRequest) qui contient le hash du fichier à signer(bloc-info.txt). La commande openssl donne ceci

```
1 openssl ts -query -data ../Dossier/tmp/bloc_info.txt -no_nonce -sha512 -
  cert -out ../Dossier/tmp/file.tsq
```

- envoyer la demande à l'horodateur par une commande curl et récupérer le timestamp.

```
1 curl -H "Content-Type: application/timestamp-query" --data-binary "@../
  Dossier/tmp/file.tsq" https://freetsa.org/tsr > ../Dossier/tmp/file.tsr
```

## V.II Constitution du QRCode

Comme nous l'avons mentionnés, le QRCode est constitué de la signature (convertie en base64) du bloc d'information et de la date de demande. Nous avons aussi ajouté la taille de la stégano pour l'utiliser dans la vérification.

## V.III La signature :

Pour réaliser la signature du bloc d'information et de la date de la demande, nous avons utilisé l'algorithme de signature sha256 et la clé privée du certificat de signature d'attestation. La date est au format **H :M :S :d :m :y**.

Voici la commande openssl utilisée pour signer , où filename est le fichier de la signature :

```
1 openssl dgst -sign ../les_cles/cle_privee_certifplus_signature.pem -keyform PEM -
  sha256 -out ../Dossier/tmp/bloc_signed.sign -binary %s'%filename
```

Sa conversion en base64 :

```
1 openssl base64 -in ../Dossier/tmp/bloc_signed.sign -out ../Dossier/tmp/
  bloc_signed_ascii.sign | cat ../Dossier/tmp/bloc_signed_ascii.sign
```

Après avoir récupéré les informations nécessaires, c'est-à-dire la signature convertie en bas64 et la taille de la stégano, nous allons les concaténés, les mettre dans l'image du QRCode et le redimensionner à l'aide de la bibliothèque mogrify.

```
1 def faire_qr_code(info,taille_stegano):
2     data = str(info)+"**"+str(taille_stegano)
3     #print(data)
4     nom_fichier = "../Dossier/tmp/qrcode.png"
5     qr = qrcode.make(data)
6     qr.save(nom_fichier, scale=2)

1 def merge_qrcode_wit_img():
2     cmd = subprocess.Popen("mogrify -resize 150x150 ../Dossier/tmp/qrcode.png",
3                             shell=True,stdout=subprocess.PIPE)
4     (resultat, ignorer) = cmd.communicate()
5     cmd = subprocess.Popen("composite -geometry +1450+1000 ../Dossier/tmp/qrcode.png
6                             ../Dossier/tmp/stegano_attestation.png ../Dossier/tmp/attestation.png", shell=
7     True,stdout=subprocess.PIPE)
8     (resultat, ignorer) = cmd.communicate()
```

## VI Vérification d'une attestation

Le processus de vérification comporte deux étapes essentielles : la vérification du timestamp et celle de la signature.

### VI.I Vérification du timestamp

Les données insérées dans la stégano sont constituées du bloc d'information + des caractères sur 64 octets + le timestamp convertit en base64. Nous allons récupérer indépendamment chacun de ces éléments. Pour réaliser la vérification du timestamp, nous avons besoin du fichier .tsq et du fichier .tsr mais alors que notre stégano ne contient que la conversion en base64 du fichier .tsr. Pour ce faire, nous allons enlever les caractères qui ont permis d'avoir les 64 octets et créer un nouveau fichier .tsq de vérification avec la commande :

```
1 openssl ts -query -data ../Dossier/tmp/bloc_info.txt -no_nonce -sha512 -cert -out
  ../Dossier/tmp/file_v.tsq
```

et vérifier si le nouveau fichier tsq a permis la création du fichier .tsr récupéré dans la stégano (en gros vérifier que ce fichier tsq est le même que celui qui a permis la création de l'attestation) avec la commande :

```
1 openssl ts -verify -in ../Dossier/tmp/file_v.tsr -queryfile ../Dossier/tmp/file_v.
  tsq -CAfile ../freeTSA/cacert.pem -untrusted ../freeTSA/tsa.crt
```



## VI.II Vérification de la signature

Nous rappelons que la signature est contenue dans le QRCode. Pour cela, nous allons récupérer le QRCode de l'image fournit par l'utilisateur et en ressortir l'information cachée à l'aide de la bibliothèque zbarlight.

```
1 def get_qrcode_from_img(fileName):
2     """
3     """
4     attestation = Image.open(fileName)
5     qrImage = attestation.crop((1450,1000,1450+150,1000+150))
6     qrImage.save("../Dossier/tmp/qrcode.png", "PNG")
7
8 def get_info_from_qrcode():
9     """
10    """
11    ....
12    image = Image.open("../Dossier/tmp/qrcode.png")
13    data = zbarlight.scan_codes(['qrcode'], image)
14    return data
```

Pour vérifier si la conformité signature, nous allons utiliser la clé publique du certificat de signature, la signature récupérée du QRCode, le bloc info et la date. Voici la commande openssl :

```
1 openssl dgst -verify ../les_cles/cle_pub_certifplus_signature.pem -keyform PEM -
   sha256 -signature %s -binary %s"%(f_name,f)
```

## VII Certificat du serveur Frontal

Nous avons vu plus haut que nos clients avant de nous envoyer une demande d'attestation ou de vérification passent par le serveur frontal. Afin d'établir une connexion sécurisée basée sur la confiance du tiers (l'AC racine CertifPlus), nous allons établir le certificat du serveur signé par l'AC racine.

### VII.I Création de la clé privée du serveur

```
1 # generer bicl pour le crtificat du serveur frontal
2 openssl ecparam -out cle_prive_serveur_frontal.pem -name prime256v1 -genkey
```

### VII.II Création du fichier .csr

```
1 openssl req -config <(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_prive_serveur_frontal.pem -text -out
   ecc.serveur.frontal.csr.pem
```

## VII.III Signature et création du bundle

```
1 # signer le certificat par l'ac
2 openssl x509 -req -days 365 -CA ca.root.cert.pem -CAkey c_p_ac.pem -CAcreateserial -
  extfile <(printf "basicConstraints=critical,CA:FALSE") -in ecc.serveur.frontal.
  csr.pem -text -out certificat_serveur_frontal.pem
3 #####
4 cat cle_prive_serveur_frontal.pem certificat_serveur_frontal.pem > bundle_serveur.
  pem
```

## VIII Lancement du programme

### \* Le serveur d'application

Pour lancer le serveur d'application, on se positionne dans la répertoire /source/CertifPlus\_AC/tools et on exécute la commande :

```
1 python3 webService.py
```

### \* Le serveur frontal

Pour lancer le serveur frontal, on se positionne dans la répertoire /source/serveur-frontal et on exécute la commande :

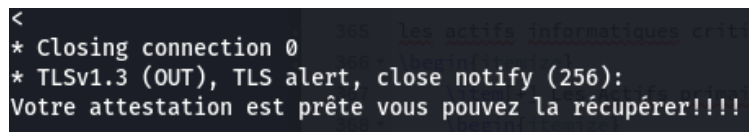
```
1 ./lancer_serveur_frontal
```

### \* Le Client

Pour lancer la demande d'attestation, on se positionne dans la répertoire /source/client et on exécute la commande :

```
1 ./request_attestation <nom> <prenom> <intitule certificat>
2 #exemple
3 ./request_attestation Toto Toto Certificat
```

Une fois l'attestation prête, le client reçoit un message l'invitant à récupérer son attestation Toujours



```
<
* Closing connection 0
* TLSv1.3 (OUT), TLS alert, close notify (256):
Votre attestation est prête vous pouvez la récupérer!!!!
```

FIGURE 4 – Message indiquant que l'attestation est prête

dans le même répertoire, le client lance la commande suivante pour récupérer l'image

```
1 ./recuperer_attestation <nom_image.png>
2 #exemple
3 ./recuperer_attestation mon_attestation.png
```

### La vérification

Toujours dans le répertoire ..., le client lance la commande suivante pour faire vérifier son attestation

```
1 ./verify_attestation <nom_image.png>
2 #exemple
3 ./verify_attestation mon_attestation.png
```

## IX Résultat :

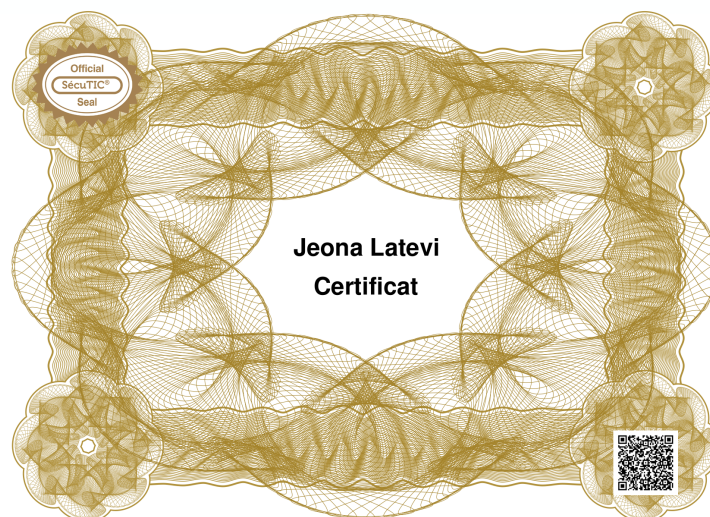


FIGURE 5 – Attestation

## X Analyse de risques

Les actifs informatiques critiques dans notre organisation sont :

- + Les Actifs primaires
  - Le serveur d'application
  - La clé privée utilisée
  - La clé privée du certificat racine
- + Les Actifs secondaires
  - Le serveur frontal
  - La clé privée du serveur frontal

Les principaux processus métier qui utilisent ces informations sont :

- délivrer et vérifier une attestation
- créer un certificat qui permet la signature d'une attestation
- créer un certificat pour se connecter au serveur frontal

Les menaces qui peuvent perturber ces fonctions métiers sont :

- Une attaque sur le serveur frontal peut perturber la délivrance et la vérification des attestations. Quelques attaques : (DDOS,XSS,SSRF,etc)
- Si un attaquant récupère la clé privée permettant de signer les attestations, il peut alors délivrer des attestations se faisant passer pour CertifPlus.
- si un attaquant récupère la clé privée permettant de gérer le certificat du serveur frontal, il peut alors faire un certificat qui redirectionne les demandes des attestations vers un serveur

- d'application malveillant.
- Étant donné que ce sont des personnes qui travaillent dans la société, les menaces peuvent venir d'un humain (erreur ou intentionnel)

## **XI   Conclusion**

Ce projet nous a permis de mettre en place un système de délivrance d'attestations de manière sécurisée par le biais de la bibliothèque opensource openssl. Nous avons manipuler en long et en large cette bibliothèque pour mener à bien ce projet. Cependant comme tout système informatique le risque zéro n'existant pas, ce système sécurisé peut être menacer.