UNIVERSITY OF LIMOGES

Faculty of Science and Technology

Master 1 CRYPTIS

Sécurité de l'Information et Cryptologie

(CRYPTIS)

Parcours Informatique

Projet Sécurité des usages TIC - Semestre II

Authentification, Web sécurisé Stéganographie

MAKHOUL Vladimir

SALAME Joe

Enseignants

M. Conchon Emmanuel

M. Bonnefoi Pierre-Francois

15 mai 2023

Table des matières

Présentation

Une société de certification CertifPlus nous a contactés afin que nous mettions en place leur procédé de diffusion électronique sécurisée d'attestations de réussite aux certifications qu'ils délivrent. Nous devons assurer l'authenticité des attestations délivrées de manière électronique, sous la forme d'une image. Cette image comprend des informations visibles telles que le nom de la personne qui reçoit l'attestation de réussite, le nom de la certification réussie et un QR code contenant la signature de ces informations.

Chapitre 1

Création de l'AC

Dans le domaine de la sécurité des certifications, la création d'une Autorité de Certification (AC) joue un rôle essentiel. Une AC bien configurée permet de délivrer des certificats utilisant les Courbes Elliptiques, une technique de cryptographie efficace et robuste.

Pour mettre en place une AC utilisant les Courbes Elliptiques, nous avons utilisé les commandes suivantes :

```
#!/bin/bash
openssl ecparam -out ecc.ca.key.pem -name prime256v1 -genkey

openssl req -config <(printf "[req]\ndistinguished_name=dn\n[dn]\n[ext]\
    nbasicConstraints=CA:TRUE\nkeyUsage=digitalSignature") -new -nodes -subj "/C
    =FR/L=Limoges/O=CertifPlus/OU=SecuTIC/CN=localhost" -x509 -extensions ext -
    sha256 -key ecc.ca.key.pem -text -out ecc.ca.cert.pem

cat ecc.ca.key.pem ecc.ca.cert.pem > bundle_serveur.pem
```

la première commande du script permet de génèrer une clé privée (ecc.ca.key.pem) en utilisant la courbe elliptique prime256v1 qui sera utilisée pour signer le certificat.

La deuxieme commande crée le certificat(self signed) spécifiant l'utilisation de la clé publique pour la signature numérique. il contient également des informations telles que le nom commun (CN=localhost) et l'organisation (O=CertifPlus).Le paramètre "basicConstraints=CA :TRUE" est utilisé pour indiquer que le certificat doit être traité comme un certificat CA. Cela garantit que le certificat peut être utilisé pour émettre et signer d'autres certificats dans une chaîne de certificats.

la troisième commande concatine la clé privée de l'AC avec la certification générée par cet AC dans un fichier ($bundle_serveur.pem$) à utiliser ultérieurement pour la communication TLS.

Chapitre 2

Création de l'attestation

2.1 Partie Client

Tout d'abord, l'étudiant envoie une demande de création d'attestation au serveur à l'aide de la commande suivante :

```
curl -X POST -d 'identite=toto' -d 'intitule_certif=SecuTIC' \ http://localhost
    :8080/creation
```

Cette commande contient les informations nécessaires à la création de l'attestation qui sont le nom de l'étudiant ('identite=toto') et le titre de l'attestation ($'intitule_certif = SecuTIC'$) (Cette commande changera lors de la configuration du serveur tls)

2.2 Partie Serveur

Récupération de la demande

Au départ le serveur reçoit la demande de l'étudiant puis il extrait le nom de ce dernier et le titre de l'attestation. Ensuite il enregistre dans une variable "temps" qui représente le temps de réception de la demande qui sera signée auprès de l'autorité d'horodatage. Ces informations seront concaténées dans une variable info et à la suite enregistrée dans les fichiers suivants (time.txt et info.txt)

```
      #extract the student's identity and certificate's name
    contenu_identite = request.forms.get('identite')
    contenu_intitule_certification = request.forms.get('intitule_certif')
    #concatination of the extracted information in the variable info
    info=contenu_identite+'|'+contenu_intitule_certification
    #save the time of the reception of the request
    timestamp=time.time()
    #creating the files to save the previous information
    try:
        registerTime=open('time.txt','w')
        dataToSign=open('info.txt','w')
    except Exception as e:
        sys.exit(1)
```

```
    #save the content of info in 'info.txt'
    dataToSign.write(info)
    dataToSign.close()
    #Write the time saved in the variable 'timestamp' in the file 'time.txt'
    registerTime.write(str(timestamp))
    registerTime.close()
```

Demande d'horodatage

Premièrement cette commande crée d'abord le fichier time.tsq (TimeStampRequest), qui contient un hachage du fichier que nous voulons signer.

```
query=['openssl','ts','-query','-data','time.txt','-no_nonce','-sha512','-cert',
    '-out','time.tsq']
subprocess.run(query)
```

La deuxième commande envoie le fichier time.tsq(TimeStampRequest) à freeTSA.org et reçoit le fichier time.tsr (TimeStampResponse)

```
 getCert= ['curl','-H','Content-Type: application/timestamp-query','--data-
    binary','@time.tsq','https://freetsa.org/tsr','-o','time.tsr']
    subprocess.run(getCert)
```

Ces commands peuvent êtres trouver sur le site de freeTSA

Signature de l'AC

Dans la fonction "$AC_Signature$", le fichier info.txt est envoyé à l'AC en utilisant une communication socket sécurisée afin qu'il puisse être signé, ensuite ce fichier sera renvoyée signee au serveur. Cela sera plus détailler dans la suite.

```
    AC_certificat=b''
    AC_signature(AC_certificat)
```

Concatenation des informations pour les cacher

Dans cette partie nous avons extrait la contenue des fichiers time.tsr et time.tsq qui seront concaténé avec les informations de l'étudiant. Ensuite nous avons ajouté un padding de zéros dans le cas où la taille des informations de l'attestation sera plus petite que 64 octets

```
    #infromation to hide
    #Open then read the contents of the files 'time.tsr' and 'time.tsq'
    try:
        signed_timestamp=open('time.tsr','rb')
        request_timestampSig=open('time.tsq','rb')
    except Exception as e:
        sys.exit(1)
    #Save the content of 'time.tsq' in the variable 'req'
    req=b''
    while True:
        line=request_timestampSig.readline()
        if not line:
            break
```

```
        req+=line
    request_timestampSig.close()

    #Save the content of 'time.tsq' (timestamp signature) in 'timeStampSignature'
    timeStampSignature=b''
    while True:
        line=signed_timestamp.readline()
        if not line:
            break
        timeStampSignature+=line
    signed_timestamp.close()

    #adding padding to info in case its less the 64 bytes
    if len(info)<64:
        for i in range(len(info),65):
            info+='0'
    #concatinate all the previous info in 'hideInfo' to hide them later on
    hideInfo=info+timeStampSignature.hex()+'|'+req.hex()
```

Création du QRcode

Pour cette partie nous avons converti le contenu de sign.sig (contenant les informations signe par le CA) en base 64 pour le convertir ensuite en QRcode

```
    #Converting the CA's signature to Base64
    base64=subprocess.Popen('openssl base64 -in sign.sig -out sign.txt',shell=True,stdout=subprocess.PIPE)
    base64.communicate()
    #QRCODE Creation
    try:
        toQr=open('sign.txt','r')
    except Exception as e:
        sys.exit(1)
    line=''
    while 1:
        l=toQr.readline().strip('\n')
        if not l:
            break
        line+=l
    my_file='qrcode.png'
    qr=qrcode.make(line)
    qr.save(my_file,scale=2)
    toQr.close()
```

Stéganographie

Dans cette section, nous avons téléchargé le fond et redimensionné les images du texte et du QRCode, ensuite nous les avons combinés pour créer l'attestation.Finalement nous avons cache les informations contenus dan la variable "hideInfo" dans limage à l'aide de la fonction cacher() déjà donner.

Envoi de l'attestation a l'étudiant

Après avoir créé l'attestation, nous spécifions le type de contenu de la réponse HTTP pour qu'il soit "image/png" Puis nous lirons l'image et nous l'enregistrons dans la variable 'image' pour l'envoyer plus tard a l'étudiant.

```
    #specify the content type to image/png
    response.set_header('Content-type', 'image/png')
    #read the image content
    try:
        img=open('final_img.png','rb')
    except Exception as e:
        sys.exit(1)
    image=b''
    while 1:
        line=img.readline()
        if not line:
            break
        #save the content in the variable 'image'
        image+=line
    img.close()
    #return the certification
    return image
```

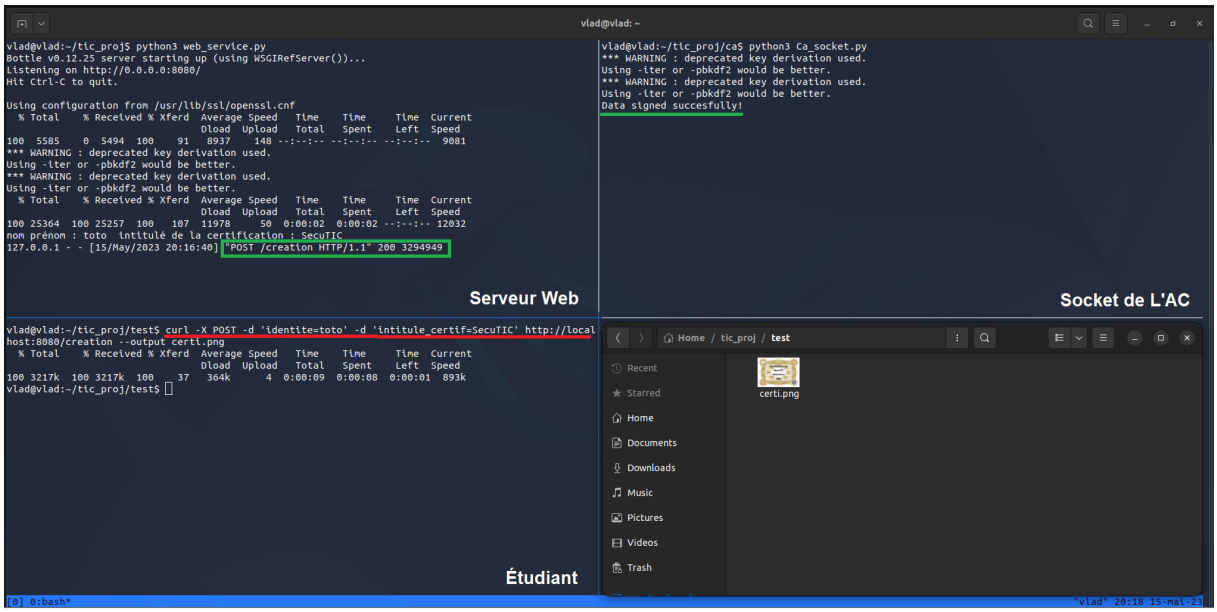
2.3 Communication socket entre l'AC et le serveur

Dans la fonction $AC_signature$, le serveur Web possède une passphrase 'secret' utilisée pour le chiffrement symétrique AES. Le serveur Web utilise d'abord cette passphrase pour chiffrer le fichier 'info.txt' avec AES qui contient les informations à signer par l'AC. Lorsque le $serveur_web$ se connecte au socket de l'AC, il reçoit le certificat de l'AC et en extrait la clé publique qui sera utilisée ultérieurement pour la vérification de la signature. Après avoir envoyé les informations chiffrées, le serveur Web recevra les informations signées et chiffrées pour ajouter une couche supplémentaire de sécurité. Le $serveur_web$ déchiffrera la signature et l'enregistrera dans le fichier 'sign.sig' qui sera utilisé pour le QRcode.

l'AC a également la même passphrase pour le chiffrement symétrique AES. Il va d'abord ouvrir le fichier du certificat pour le lire et l'enregistrer dans la variable 'certificat'. Lorsque le socket de l'AC acceptera une connexion il enverra le certificat puis il recevra les informations chiffrée et les enregistrera dans 'information.AES'. Après avoir déchiffré les informations, l'AC les signera à l'aide de sa clé privée. Enfin, l'AC chiffrera la signature pour la renvoyer au web_server **Le code du socket de l'AC se trouve dans le repertoir ac sous le nom de Ca**$_socket$

2.4 Exécution

Comme nous pouvons le voir sur l'image, l'étudiant a envoyé la demande au serveur qui a obtenu ces informations et les a envoyées à l'aide de la communication socket à l'AC pour qu'elles soient signées.Nous pouvons effectivement voir l'attestation dans le répertoire ou l'étudiant a fait la requête



Chapitre 3

Vérification de l'attestation

Extraction et lecture du QRcode

Dans cette partie, nous recevons l'attestation de l'étudiant, puis nous en extrayons le Qrcode et nous le lisons à l'aide de la bibliothèque "zbar". Ensuite, nous le décodons à partir de base64 vue que nous l'avons encodé avant de créer le QRcode

```
    def v rification_attestation():
    response.set_header('Content-type', 'text/plain')
    contenu_image = request.files.get('image')
    contenu_image.save('attestation_a_verifier.png',overwrite=True)
    #get the QrCode
    attestation = Image.open('attestation_a_verifier.png')
    qrImage = attestation.crop((1418,934,1418+210,934+210))
    qrImage.save("qrcoderecupere.png", "PNG")
    #Read the QrCode
    image = Image.open("qrcoderecupere.png")
    data64 = zbarlight.scan_codes(['qrcode'], image)
    data64 = data64[0]
    try:
        Qrout = open('qrout64.txt','w')
    except Exception as e:
        sys.exit(1)
    Qrout.write(data64.decode()+"\n")
    Qrout.close()
```

Extraction des informations

Maintenant nous allons extraire les informations de la photo en utilisant la fonction de stéganographie "récupérateur", supprimer le remplissage puis enregistrer les informations dans 3 fichiers, un pour les identifiants des étudiants, un pour la signature d'horodatage et un pour la demande d'horodatage.

```
    try:
        user_information=open('verify/infoToCheck.txt','w')
        timeStampSig=open('verify/timeStampSig.txt','w')
        timeStampReq=open('verify/timeStampReq.txt','w')
    except Exception as e:
        sys.exit(1)
```

```
    hidden_info = recuperer(attestation, 64+10986+1+182)
    info=''
    for i in hidden_info:
        if i!='0':
            info+=i
        else:
            break
    #save user info
    user_information.write(info)
    user_information.close()
    timeStampInfos=hidden_info[65:].split('|')
    #save timeStamp signature
    timeStampSignature=timeStampInfos[0]
    timeStampSig.write(timeStampSignature+'\n')
    timeStampSig.close()
    #save timeStamp request
    timeStampRequest=timeStampInfos[1]
    timeStampReq.write(timeStampRequest+'\n')
    timeStampReq.close()
```

Vérification de la signature des informations de l'étudiant

La commande de vérification de la signature utilise l'algorithme de hachage SHA-256 et spécifie la clé publique de l'autorité de certification (AC) à utiliser pour la vérification. Elle prend également en entrée le fichier "infoToCheck.txt" contenant les informations à vérifier et le fichier de signature "sign.sig".

```
    veri_commande = "openssl dgst -sha256 -verify keys/AC_public_key.pem -signature sign.sig verify/infoToCheck.txt"
    veri_commande1 = subprocess.Popen(veri_commande,shell=True,stdout=subprocess.PIPE, stderr=subprocess.PIPE)
    stdout,stderr=veri_commande1.communicate()
```

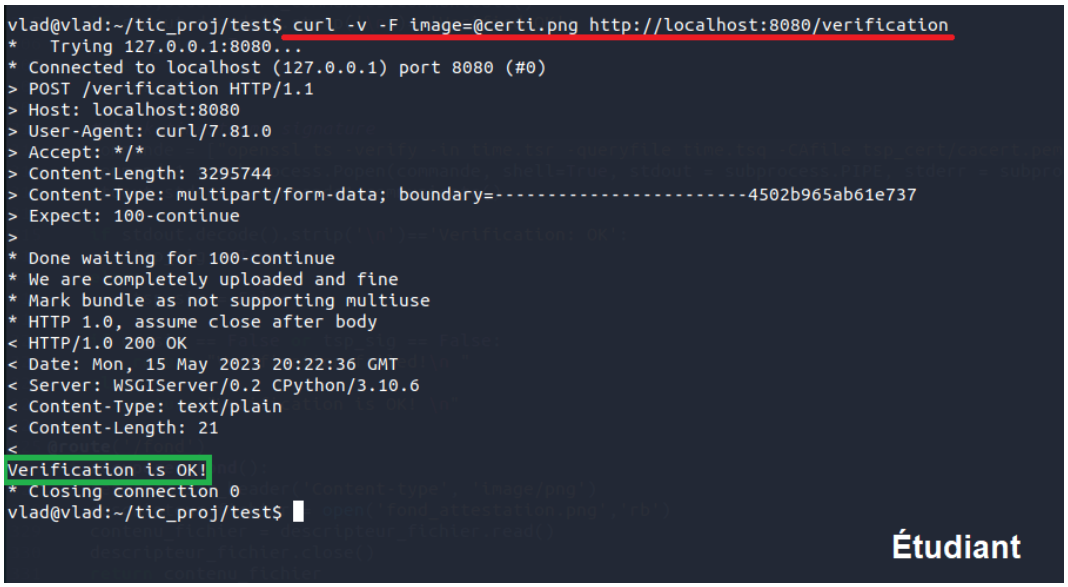
Vérification de la signature de l'horodatage

La commande de vérification de l'horodatage utilise les fichiers "time.tsr" et "time.tsq" pour la vérification. Elle spécifie également les fichiers de certificat à utiliser pour la vérification, y compris le certificat de l'autorité de certification (CA) et un certificat d'autorité de signature de temps (TSA).

```
    commande = ["openssl ts -verify -in time.tsr -queryfile time.tsq -CAfile tsp_cert/cacert.pem" ,"-untrusted" , "tsp_cert/tsa.crt" ]
    commande1 = subprocess.Popen(commande, shell=True, stdout = subprocess.PIPE, stderr = subprocess.PIPE)
    stdout,stderr = commande1.communicate()
```

Exécution

Nous pouvons bien voir qu'après l'exécution de la commande de vérification, l'attestation a été valider vue que personne n'a altéré son contenu.



Chapitre 4

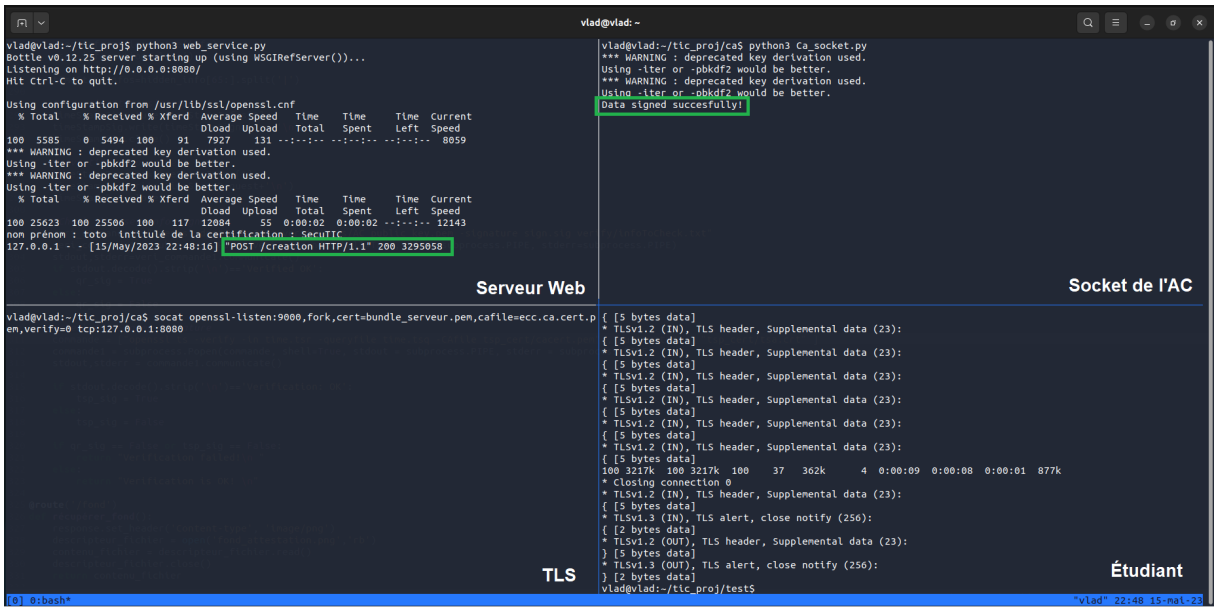
Communication TLS

Maintenant, pour configurer la connexion TLS, nous avons d'abord concaténé la clé privée de l'AC avec le certificat dans un fichier $'bundle_serveur.pem'$ qui sera utilisé dans la commande socat. La commande socat utilise le fichier créé précédemment $'cert = bundle_serveur.pem'$ pour chiffrer et déchiffrer les données. L'option 'cafile=ecc.ca.cert.pem' permet à l'utilisateur de vérifier l'identité de l'AC. Enfin, nous avons ajouté à la commande curl l'option –cacert qui prend le certificat à utiliser pour la vérification d'identité.

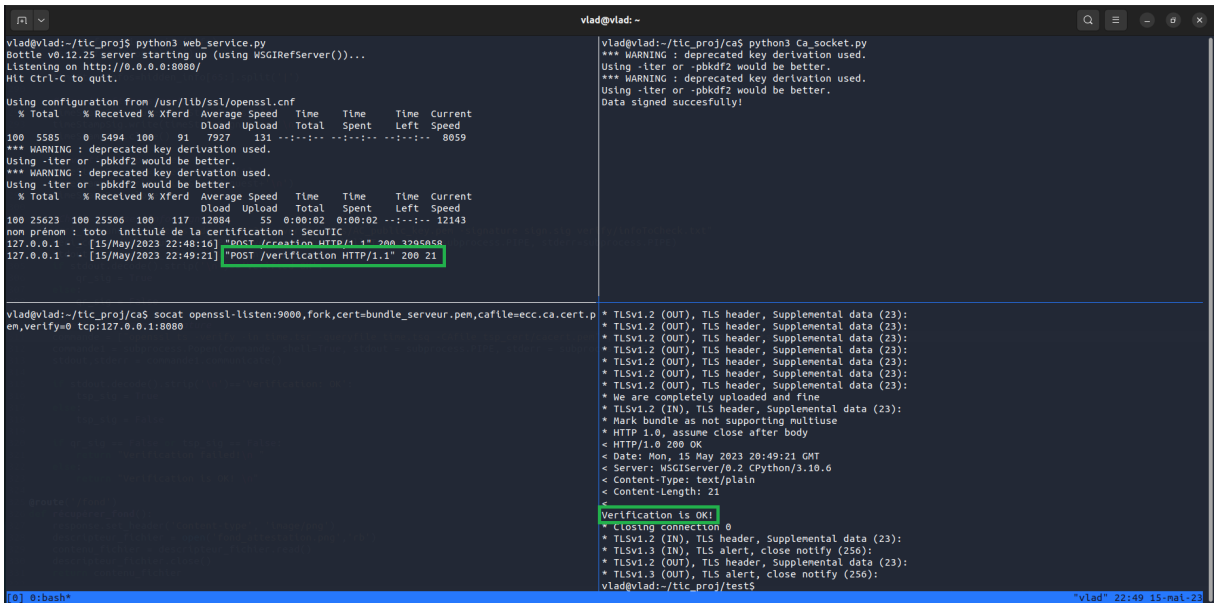
```
1 socat openssl-listen:9000,fork,cert=bundle_serveur.pem,cafile=ecc.ca.cert.pem,
    verify=0 tcp:127.0.0.1:8080
2
3 curl -v -X POST -d 'identite=toto' -d 'intitule_certif=SecuTIC' --cacert ../ca/
    ecc.ca.cert.pem https://localhost:9000/creation --output 'certi.png'
4
5 curl -v -F image=@certi.png --cacert ../ca/ecc.ca.cert.pem https://localhost
    :9000/verification
```

4.1 Exécution avec TLS

4.1.1 Création



4.1.2 Vérification



Chapitre 5

Analyse des risques

5.1 Actifs primaires

1. **Données personnelles des clients : Les informations personnelles des étudiants, telles que leur nom, le nom de la certification, doivent être protégées contre toute divulgation ou utilisation non autorisée.**
2. **Attestations : Les attestations délivrés aux étudiants doivent être protégés contre toute falsification ou altération.**
3. **Clés de chiffrement : Les clés utilisées pour chiffrer les données sensibles ou pour la génération de signatures électroniques doivent être protégées pour garantir l'intégrité et la confidentialité des informations.**

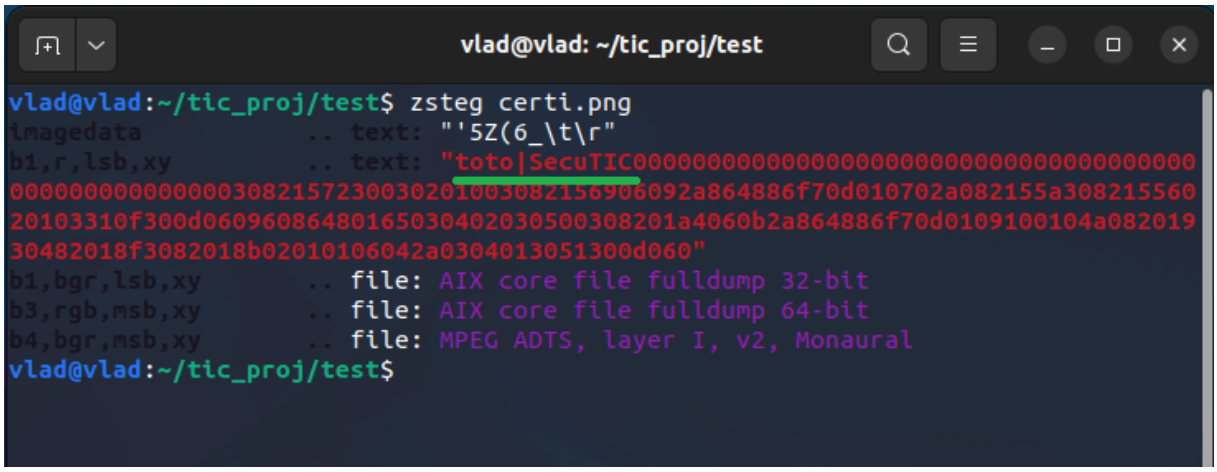
5.2 Actifs secondaires

1. **Infrastructure du serveur : Les serveurs qui hébergent l'application doivent être sécurisés contre les attaques, les intrusions et les dénis de service afin de garantir la disponibilité et la continuité du service.**
2. **Certificats d'autorité : Les certificats utilisés pour la signature électronique et l'authentification doivent être protégés pour éviter les abus ou les compromissions.**

5.3 Les menaces sur ces biens

5.3.1 Confidentialité des données

Un risque potentiel est la divulgation non autorisée des données du client pendant la transmission entre le client et le serveur, ainsi que pendant l'envoi de l'attestation à FreeTSA. Cela peut compromettre la confidentialité des informations personnelles du client, telles que son nom, ses certifications et d'autres données sensibles.
En effet la confidentialité des donnes n'a pas été atteinte ici, comme vous le voyez nous avons utilisé l'outil "zsteg" qui est l'un des outils les plus puissants et connu en steganography, et nous avons pu extraire les informations de l'étudiant de l'attestation.



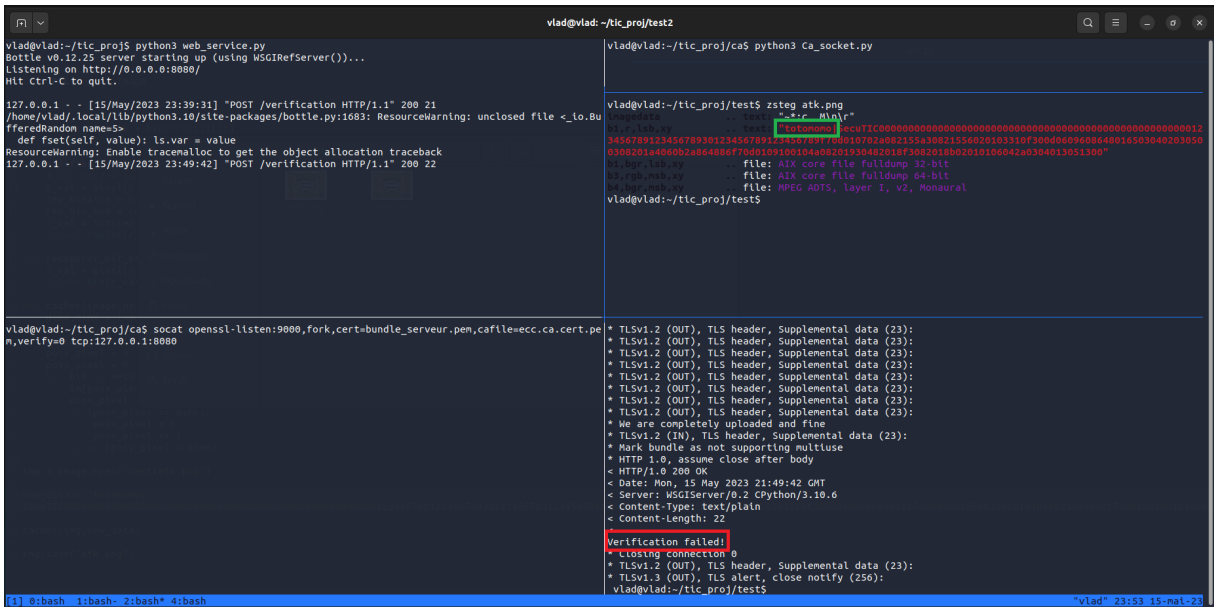
Solutions

- **Utiliser un algorithme de stéganographie robuste**
- **Chiffrer les données sensibles avec un algorithme de chiffrement avant de les dissimuler dans la photo**

5.3.2 Intégrité des données

Il existe un risque de modification non autorisée des données pendant la transmission entre le client et le serveur, ainsi que pendant la signature de l'attestation par FreeTSA. Si les données sont altérées de manière malveillante ou accidentelle, cela peut remettre en question l'intégrité de l'attestation et de ses informations.

Dans cet exemple, nous avons modifié les informations à l'intérieur de la photo à l'aide de l'outil "Steghide", comme vous pouvez le voir, les données ont changé. Mais dans ce cas l'intégrité des données est atteinte vue que la vérification n'a pas été valider par le serveur, donc le client va savoir que ses données ont été altérer



5.3.3 Sécurité de la signature électronique

L'utilisation d'une autorité de certification (CA) locale pour signer le QR code contenant les informations du client présente des risques. Si la sécurité de la clé privée de la CA est compromise, cela pourrait permettre à un attaquant de générer de fausses attestations avec des informations modifiées ou non valides.

5.3.4 Sécurité du canal de communication

L'utilisation du protocole TLS pour la communication entre le client et le serveur est une mesure de sécurité importante. Cependant, des vulnérabilités dans la configuration de TLS ou des attaques telles que les attaques de type Man-in-the-Middle peuvent compromettre la confidentialité et l'intégrité des données transmises. Bien que l'utilisation de TLS contribue à sécuriser les communications en chiffrant les données, il existe des scénarios où un attaquant peut encore mener une attaque MITM. Voici quelques exemples de ces scénarios :

1. **Certificat frauduleux : L'attaquant peut générer un certificat SSL/TLS frauduleux et se faire passer pour le serveur légitime. Si le client accepte le certificat frauduleux, l'attaquant peut intercepter et déchiffrer les communications entre le client et le serveur.**
2. **Attaque de renégociation TLS : Certaines implémentations de TLS antérieures à TLS 1.3 peuvent être vulnérables à des attaques de renégociation, où un attaquant peut exploiter les négociations de clés pour s'insérer dans la communication.**
3. **Attaque de downgrade : Un attaquant peut tenter de forcer la communication entre le client et le serveur à utiliser une version plus faible de TLS ou un protocole non sécurisé, rendant les données vulnérables à l'interception.**
4. **Attaque contre les certificats de l'autorité de certification (CA) : Si l'attaquant parvient à compromettre une autorité de certification de confiance ou à obtenir un certificat valide frauduleux, il peut l'utiliser pour mener une attaque MITM et tromper le client.**

Solutions

1. **Mettre en œuvre la vérification d'intégrité des certificats, telle que la vérification de l'empreinte digitale (fingerprint) du certificat.**
2. **Utiliser la version la plus récente de TLS et mettre à jour régulièrement les logiciels pour bénéficier des correctifs de sécurité.**
3. **Sensibiliser les utilisateurs aux bonnes pratiques de sécurité, comme ne pas ignorer les avertissements de certificat non valide et ne pas accéder à des sites sensibles via des connexions non sécurisées**