List of Tables

List of Figures

Contents

I. Introduction

In this project, we are suppose to implement a distribution processsecure electronic certificate of success for CertifPlus company.

Objectives

- An user can:
    - request to create a certificate with their information
    - download their certificate
    - verify an existing certificate
- The authenticity of the certificate issued electronically in the form of an image must be guaranteed:
    - The image contains visible information :
        - The name of the person receiving the certificate of achievement
        - The name of the successful certification
        - A QRcode containing the signature of this information
    - The image contains hidden information :
        - tamper-proof information is concealed by steganography in the image. This information includes the visible information of the certificate as well as the guaranteed delivery dateby a *timestamp* signed by a time stamping authority **freetsa**
- Verification
    - extract and the stamp concealed in the image by steganography and verify timestamp
    - checks the signature encoded in the QRcode

II. Programs, Materials, Methodologies

2.1 Programs and Materials

- Python 3.8
- bottle - For Web Services
- qrcode, numpy, Pillow, zbarlight for qrCode creation, verification and image modification
- a stenography library provided in this project
- socat - multipurpose relay tool

2.2 Methodologies

A. Creating certficate

An user will request CertifPlus for creating a certificate by providing his/her informations containing Last name and First name, Institue.

First of all, we sign their signature using `SHA-256` with binary file output and using encrypted CA key by using the inforBlock containing the user name and the institute which is extended as 64 characters string.

```
$ openssl dgst -sha256 -sign CA/enc.ecc.ca.key.pem ./CA/info.txt
    > ./CA/signature.sig
```

This signature after that will be used to create a `TimeStampRequest` file, which contains a hash of this signature file that we want to sign.

```
$ openssl ts -query -data file.png -no_nonce -sha512 -cert -out
    file.tsq
```

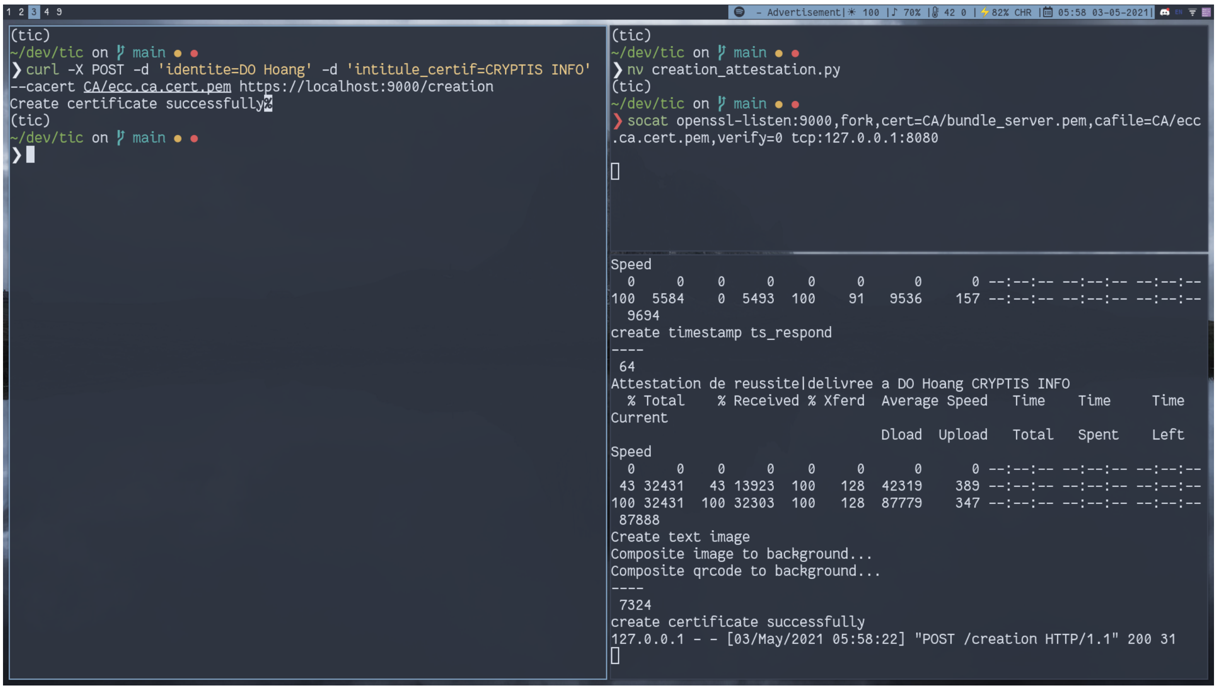
Then we send the `TimeStampRequest` file to *freeTSA* and receive a `TimeStampResponse` file which is going to be used as a stegano message later.

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

We also use that signed signature to create a QRcode containing this informations and then combining the text image that received by using google chart API with their `inforBlock` that we mentioned earlier, QRcode and attestation font provided in this project.

The stegano message is contained the infor block and `TimeStampResponse` which is converted to *ascii.* Finally, the certificate will be create.

Result


1 2 3 4 9
- Advertisement|☀ 100 |♪ 70% |42 0 |82% CHR |05:58 03-05-2021|
EN

```
(tic)
~/dev/tic on main ● ●
❯ curl -X POST -d 'identite=DO Hoang' -d 'intitule_certif=CRYPTIS INFO'
--cacert CA/ecc.ca.cert.pem https://localhost:9000/creation
Create certificate successfully%
(tic)
~/dev/tic on main ● ●
❯
```

```
(tic)
~/dev/tic on main ● ●
❯ nv creation_attestation.py
(tic)
~/dev/tic on main ● ●
❯ socat openssl-listen:9000,fork,cert=CA/bundle_server.pem,cafile=CA/ecc
.ca.cert.pem,verify=0 tcp:127.0.0.1:8080
```

```
Speed
  0      0    0      0    0      0      0      0 --:--:-- --:--:-- --:--:--
100  5584    0  5493  100     91   9536    157 --:--:-- --:--:-- --:--:--
  9694
create timestamp ts_respond
----
 64
Attestation de reussite|delivree a DO Hoang CRYPTIS INFO
  % Total    % Received % Xferd  Average Speed   Time    Time     Time
Current
                                 Dload  Upload   Total   Spent    Left
Speed
  0      0    0      0    0      0      0      0 --:--:-- --:--:-- --:--:--
 43 32431   43 13923  100    128  42319    389 --:--:-- --:--:-- --:--:--
100 32431  100 32303  100    128  87779    347 --:--:-- --:--:-- --:--:--
 87888
Create text image
Composite image to background...
Composite qrcode to background...
----
 7324
create certificate successfully
127.0.0.1 - - [03/May/2021 05:58:22] "POST /creation HTTP/1.1" 200 31
```

Figure 1: Create Certificate Successfully

B. Verify certificate

In this part, an user will request CertifPlus for verifying his/her certificate by providing his/her cert.

For verify the signature, we firstly need to extract the data from the QRcode by using `Pillow` library in Python to crop the exact QRcode from the certificate. Then using `zbarlight` to extract the data.

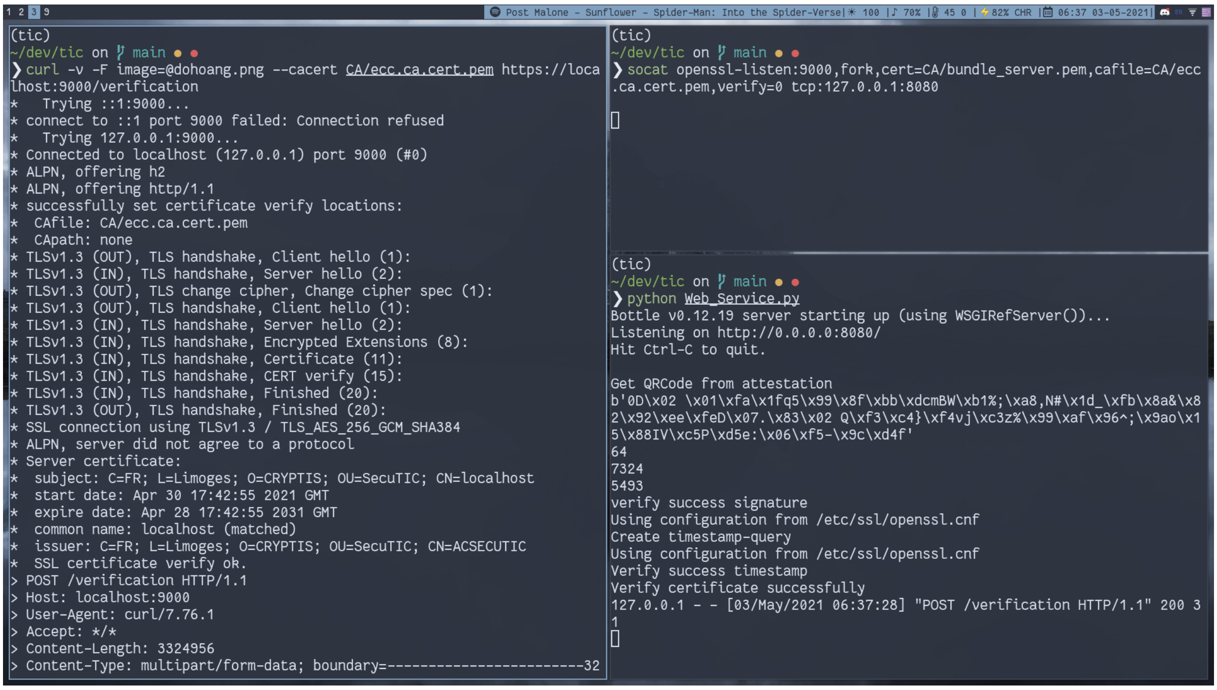
We also need to recover the message hidden in the certificate by using a stegano library provided in this project. Because we have already know the len of the message which is 64 bytes (for the infor block) and the fixed timestamp size (which is 7324) so that it is easy to extract exact the infor block and the timestamp respond.

By using the signature which is recovered from QRcode and the infor block extracted from the stegano message, we can easily verify the signature with the CA public key.

```
$ openssl dgst -verify CA/ecc.ca.pubkey.pem -signature signature.sig info.txt
```

To verify timestamp, we have already verify the signature is valid or not, we use this signature as the data for creating a `TimeStampRequest` file (same as in creating certificate part). Next, with the public Certificates(provided by *freeTSA*) you can verify the `TimeStampRespond`.

```
$ openssl ts -verify -in file.tsr -queryfile file.tsq -CAfile cacert.pem -untrusted tsa.crt
```


1 2 3 9
Post Malone - Sunflower - Spider-Man: Into the Spider-Verse| 100 | 70% | 45 0 | 82% CHR | 06:37 03-05-2021|
EN

```
(tic)
~/dev/tic on main ● ●
❯ curl -v -F image=@dohoang.png --cacert CA/ecc.ca.cert.pem https://loca
lhost:9000/verification
*   Trying ::1:9000...
* connect to ::1 port 9000 failed: Connection refused
*   Trying 127.0.0.1:9000...
* Connected to localhost (127.0.0.1) port 9000 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
* successfully set certificate verify locations:
*  CAfile: CA/ecc.ca.cert.pem
*  CApath: none
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.3 (OUT), TLS change cipher, Change cipher spec (1):
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.3 (IN), TLS handshake, Encrypted Extensions (8):
* TLSv1.3 (IN), TLS handshake, Certificate (11):
* TLSv1.3 (IN), TLS handshake, CERT verify (15):
* TLSv1.3 (IN), TLS handshake, Finished (20):
* TLSv1.3 (OUT), TLS handshake, Finished (20):
* SSL connection using TLSv1.3 / TLS_AES_256_GCM_SHA384
* ALPN, server did not agree to a protocol
* Server certificate:
*  subject: C=FR; L=Limoges; O=CRYPTIS; OU=SecuTIC; CN=localhost
*  start date: Apr 30 17:42:55 2021 GMT
*  expire date: Apr 28 17:42:55 2031 GMT
*  common name: localhost (matched)
*  issuer: C=FR; L=Limoges; O=CRYPTIS; OU=SecuTIC; CN=ACSECUTIC
*  SSL certificate verify ok.
> POST /verification HTTP/1.1
> Host: localhost:9000
> User-Agent: curl/7.76.1
> Accept: */*
> Content-Length: 3324956
> Content-Type: multipart/form-data; boundary=------------------------32
```

```
(tic)
~/dev/tic on main ● ●
❯ socat openssl-listen:9000,fork,cert=CA/bundle_server.pem,cafile=CA/ecc
.ca.cert.pem,verify=0 tcp:127.0.0.1:8080

▯
```

```
(tic)
~/dev/tic on main ● ●
❯ python Web_Service.py
Bottle v0.12.19 server starting up (using WSGIRefServer())...
Listening on http://0.0.0.0:8080/
Hit Ctrl-C to quit.

Get QRCode from attestation
b'0D\x02 \x01\xfa\x1fq5\x99\x8f\xbb\xdcmBW\xb1%;\xa8,N#\x1d_\xfb\x8a&\x8
2\x92\xee\xfeD\x07.\x83\x02 Q\xf3\xc4}\xf4vj\xc3z%\x99\xaf\x96^;\x9ao\x1
5\x88IV\xc5P\xd5e:\x06\xf5-\x9c\xd4f'
64
7324
5493
verify success signature
Using configuration from /etc/ssl/openssl.cnf
Create timestamp-query
Using configuration from /etc/ssl/openssl.cnf
Verify success timestamp
Verify certificate successfully
127.0.0.1 - - [03/May/2021 06:37:28] "POST /verification HTTP/1.1" 200 3
1
▯
```

Figure 2: Verify Certificate Successfully

III. Risk analysis

Table 1: (Risk Analysis)

Assets		Vulnerability
Primary assets		
	CA certificate	
	Server Certificate	
	ISP (internet provider)	Connection failure
	Password private key	Exposed password
Secondary assets		

Assets		Vulnerability
	FreeTSA service	TSA is not trusted
	Google chart API	Deprecated
	LAN	Connection failure

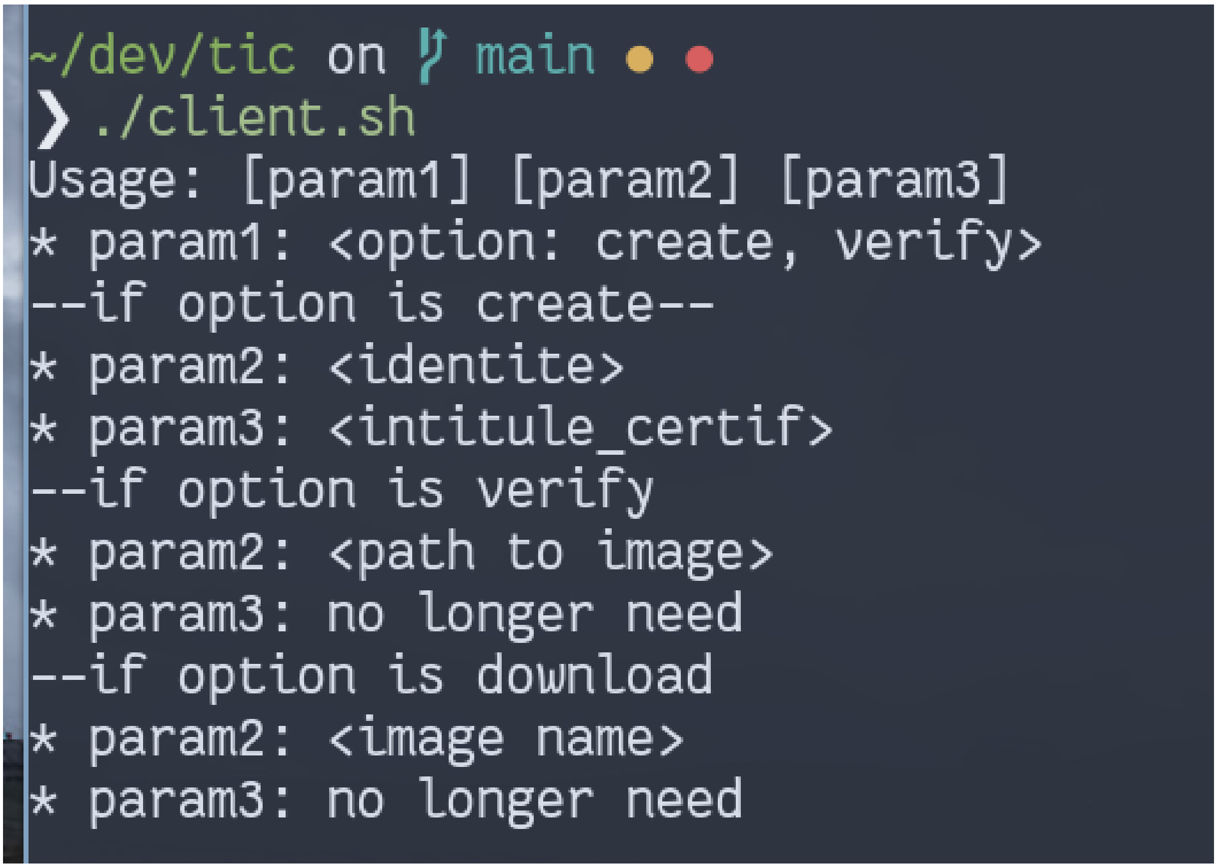
IV. Usage

Firstly, We have already prepared all the keys and certificates of CA, server and ertificates of *freetsa* with the password for encrypt private key is: `hoangphuong`

If you want to re-create all of these you can run the `init.sh` script to generate all the CA, ts and images needed in this project.

Client

We create a client script providing option for user (run `client.sh`)



```
~/dev/tic on ᚴ main ● ●
❯ ./client.sh
Usage: [param1] [param2] [param3]
* param1: <option: create, verify>
--if option is create--
* param2: <identite>
* param3: <intitule_certif>
--if option is verify
* param2: <path to image>
* param3: no longer need
--if option is download
* param2: <image name>
* param3: no longer need
```

Figure 3: Client script

Example

```
./client.sh create "Do Hoang" "Cryptis Info"
./client.sh download "dohoang.png"
./client.sh verify "dohoang.png"
```

Manually

Run web server

```
python Web_Service.py
```

Run frontal server

```
socat openssl-listen:9000,fork,cert=CA/bundle_server.pem,cafile=
    CA/ecc.ca.cert.pem,verify=0 tcp:127.0.0.1:8080
```

Create certificate

```
curl -X POST -d 'identite=DO Hoang' -d 'intitule_certif=CRYPTIS
    INFO' --cacert CA/ecc.ca.cert.pem https://localhost:9000/
    creation
```

Download the attestation and save as `dohoang.png`

```
curl -v -o dohoang.png --cacert CA/ecc.ca.cert.pem https://
    localhost:9000/fond
```

Verify the certificate

```
curl -v -F image=@dohoang.png --cacert CA/ecc.ca.cert.pem https
    ://localhost:9000/verification
```