Project 2 & 3:

Segurança Informática e nas Organizações

Diogo Moreira nmec: 93127

Fábio Carmelino nmec: 93406

Introduction

The purpose of this project is to implement security mechanisms for a media player which allows users to listen to the media from the server's catalog. With this in mind, the project was divided in two parts. The **encryption of information**, which is used to create a secure channel between the users and the server, making the communication secure, and the **authentication**, which guarantees the identity of the server as well as the client.

Workflow

1. The client asks for the **server's certificate** and confirms its validity.
2. The **client** sends his **certificate** to the server, which, likewise, gets its validity checked.

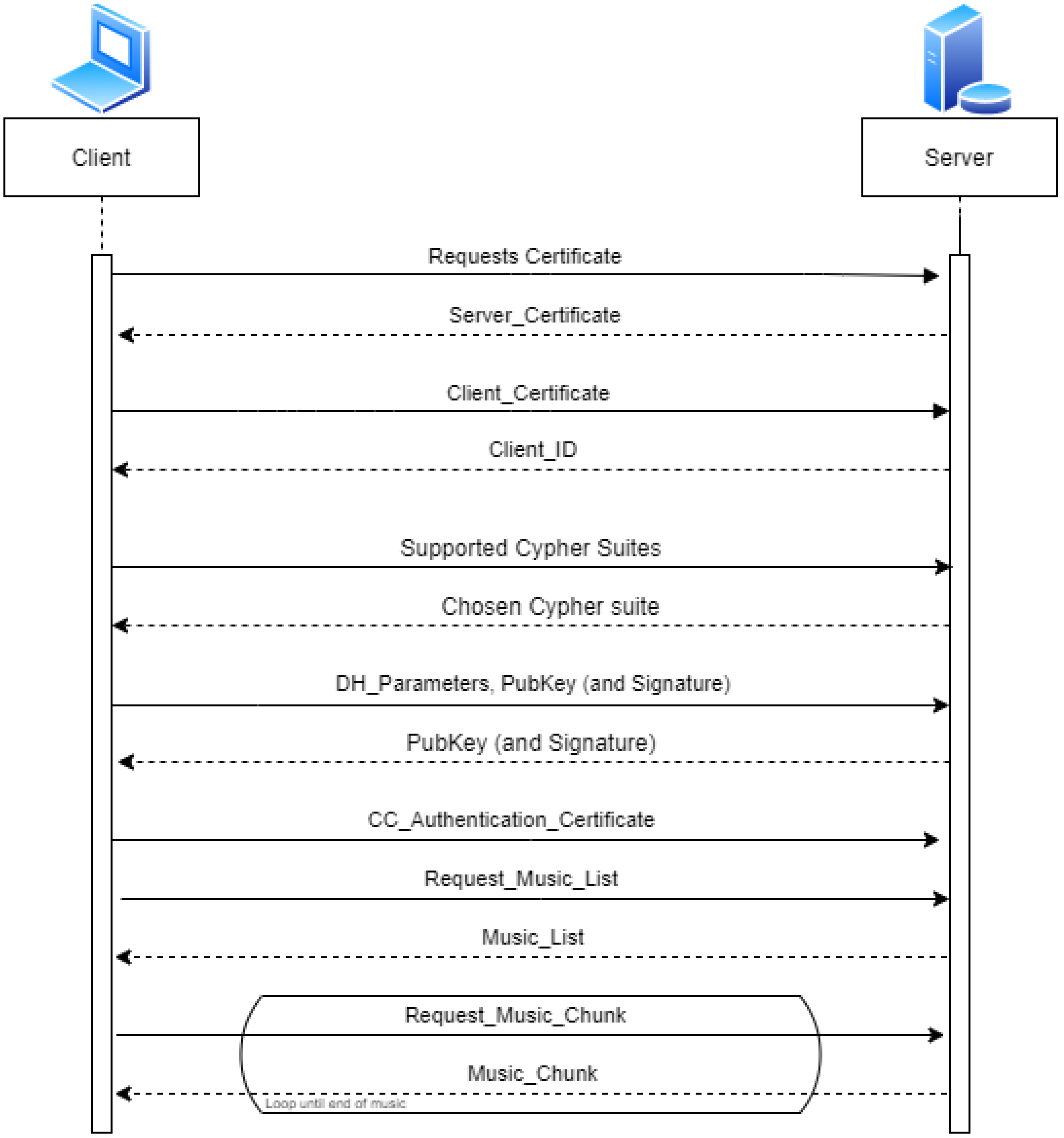
After **both** certificates are confirmed as **valid**, we start the communication:

3. Negotiation of a **cypher suite** by the parties in question.
4. Both client and server agree on a **shared key** using **Diffie-Hellman**.

From this point on, the messages can be encrypted using the shared key and the algorithms negotiated on point 3.

5. The client sends his **CC**'s (**Citizen Card**) authentication certificate.
6. The client requests the **media list** from the **server's catalog**.
7. The client can request any of the available musics and the server will create a **license for the selected music.** Every time a license for a music expires, the user has to restart the app to get a new one.

Diagram


Client
Server
Requests Certificate
Server_Certificate
Client_Certificate
Client_ID
Supported Cypher Suites
Chosen Cypher suite
DH_Parameters, PubKey (and Signature)
PubKey (and Signature)
CC_Authentication_Certificate
Request_Music_List
Music_List
Request_Music_Chunk
Music_Chunk
Loop until end of music

Implementation

1. Server's certificate

For the implementation of this request, the client sends a GET request to the server's endpoint **'/api/auth'**. On his side, the **server** will load his **certificate** and **send it to the client** who will then evaluate if he should or should not trust it. The client loads his trusted **CAs** (**Certification authorities**) and verifies if the **server's certification is signed** by any of them. In case this is true and it is still not expired, the client will choose to **trust the server**. Otherwise, the **client program will be closed**.

2. Client's certificate

The client sends a POST request to the server's endpoint **'/api/auth'.** To handle this request, the server will do the same the client did previously and in case that **the certificate can be regarded as trustful**, it will be accepted and the server will respond by sending an id for the client. This **id** will work as an **identifier** to store and access the keys and other relevant information regarding the current connection with this user.

3. Cypher Suites

The algorithms used are the following:

Symmetric cyphers:

- AES
- 3DES

Cypher Modes:

- GCM
- CBC

Digests:

- SHA256
- SHA512

With the exchange of the certificates completed, both parties know that they can **trust** each other and henceforth the next step can take place, which is the **negotiation of a cypher suite**.

This step **starts on the client side**. A POST request will be sent to the endpoint **'/api/suite'** with a list comprised of the supported cypher suites, whereas the server generates a list of the common ones, selects one randomly and sends it back to client.

4. Diffie-Hellman

Having finished negotiating the algorithms, for putting them to use and having a secure communication between server and clients, a mechanism for **generating a secret key** that will feed these algorithms is still necessary. For this task, key exchanging resorting to **Diffie-Hellman** was implemented, which allows the two parties to have a **shared key** for **symmetric encryption** to encrypt the content of the messages being exchanged.

The client generates the parameters ‘p’ and ‘g’, his **private and public key**. Both parameters (‘p’ and ‘g’) and the public key are sent to the server’s endpoint **’/api/key'** with a POST request. On his side, the server uses the parameters to **generate a private key** and a **public key** from his private key, responding to the client with his public key. At last, knowing each other's public keys, the **shared key can be generated**.

However, to prevent a **man-in-the-middle** attack, what was previously said isn’t enough. In reality, together with the DH public keys, a **signature** to **ensure the authenticity** is also sent.

In more detail, the signatures are generated by each party by **signing the DH public key with their certificate’s private key**. This implies that the other side will be able to confirm the authenticity of the DH public key. Only the **public key** of the other party’s certificate can decrypt the signature made by it and verify if **it matches** the public key sent in clear text. If it happens that the **signature is invalid**, it means that the public key was tampered with. In that way, the chance of a third-party intercepting the connection and impersonating one of the parties becomes **minimal**.

Extra Topic: Encrypted Messages

With the Diffie-Hellman concluded, a **derived key**, with its size adapted to the cypher selected in cypher suite negotiation, is **generated from the shared key** so that further exchanged messages can start being encrypted. Together with the encrypted content, an **IV**, a **hmac** and a **tag** will be sent.

The **IV** is a random number used to feed the **symmetric encryption** algorithm together with the derived key, the **tag** is necessary only in some algorithms like **3DES** and lastly the **hmac** (digest over the concatenation of the encrypted bytes and the derived key) is used to verify the **integrity of the encrypted message**.

Every time a message is sent or received, both parties run a **key rotation algorithm** which updates the derived key to a new key by performing **2 XORs**. The first is executed between the current one and the clear text of the encrypted message while the second happens between the product of the last one and the shared key.

5. Citizen's Card authentication

For this step, the client must have his **CC (Citizen's Card)** inserted on the Card Reader device. The program will load the 'Authentication Certificate' and send it via POST to the endpoint **'/api/cc'** . Receiving this request, the **server** will search his database (CCs folder) and check if there is any **certificate that matches the one sent** by the client. If there is, the server responds positively, otherwise, the response code is set to 401 and the connection is refused.

6. Media's list

The user sends a GET request to the endpoint **'/api/list'** and the server generates a list of the musics on his catalog which sends to the client.

7. Select a music

When the user selects a music, a POST request with the name of the selected music is sent to the endpoint **'/api/license'**. To handle this request, **the server will check** if there is a **folder with the serial number of this client's CC as its name**. In case there is not, one will be created. Moreover, it is necessary to verify if the **user** already has a license for this music or not.

In case the user doesn't have one, it will be generated with 2 replays left, since the current request will already count as the first time the music is played, for a total of 3 times per music.

If he has one, **the number of views available will be decremented by one** and when the number of **views reaches 0**, the next time the user requests that music, the response code will be set to 401, the **connection refused and the license file deleted.** Whereas the client will have to restart the app to get a **new license** for it.

After the license is updated, the client requests chunk by chunk, start being sent to the endpoint **'/api/download'.** And in response, the server encrypts each chunk and sends it.

Files at rest:

There are **three types** of files encrypted at rest, which were encrypted offline by executing a file named **‘protect_files.py’**. Those files are either private keys, licenses or musics. For this encryption we used **‘Fernet symmetric encryption’** and saved the used keys hardcoded on the **server and client** (the file with the client certificate’s private key is encrypted).

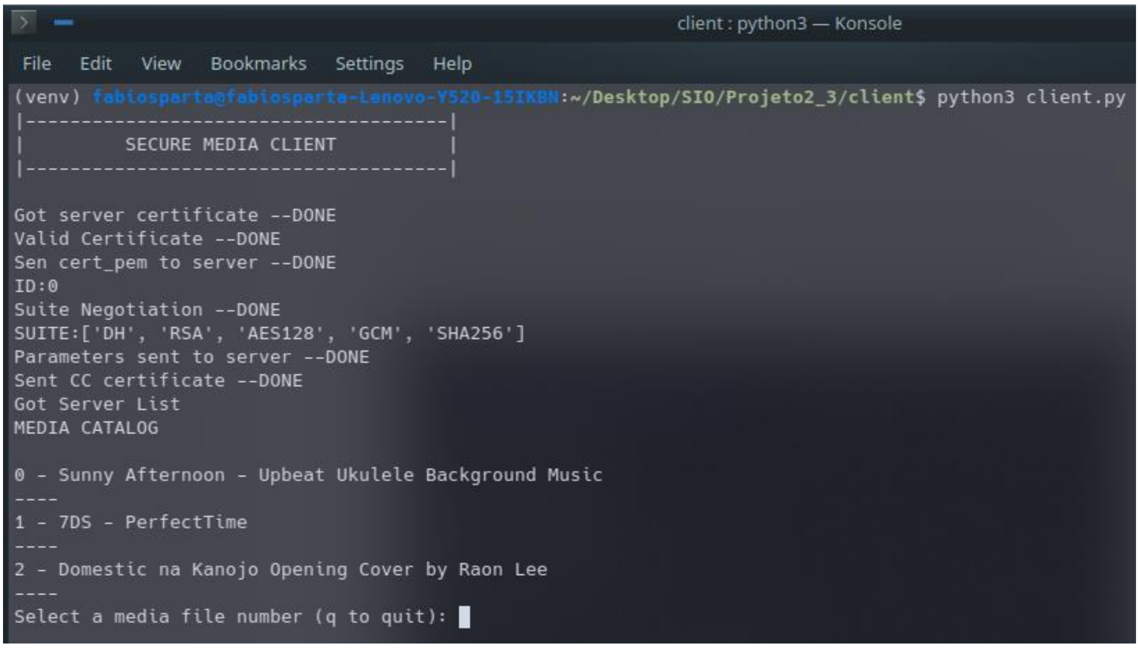
Every time a request for the **first chunk of a music** is received on the **server**, the server reads the corresponding file and **decrypts** it to a variable named *current_music* associated with **each user** connected, so that when the next chunks are requested, they are read from the byte array that **already contains the whole music decrypted**. Meaning that, the decrypted musics are never present in the disk, only in memory when necessary.

Each time a **license** is created it is already encrypted. And so, when its necessary to read it and update it a decryption is always necessary.

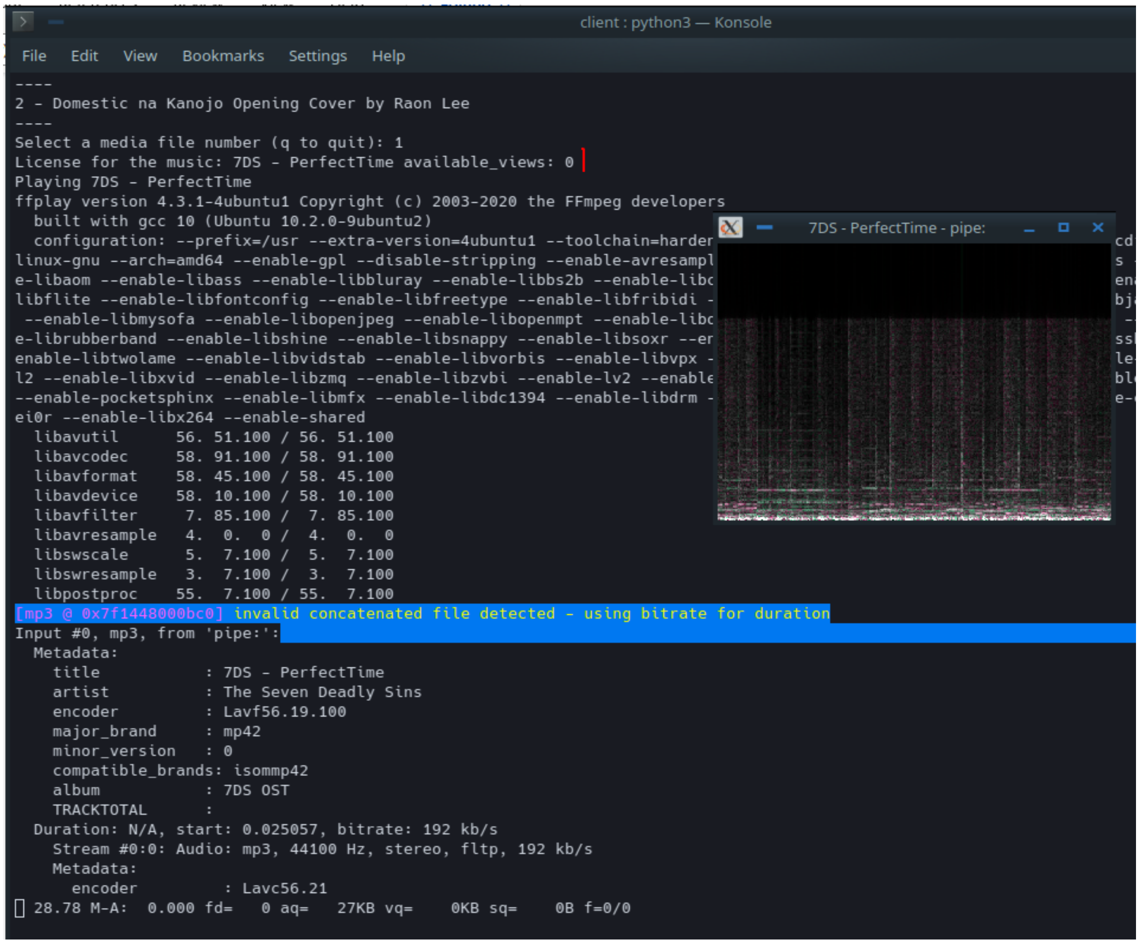
A similar situation happens with the **private keys'** correspondent to the server and user’s certificates. They must be decrypted before being used.

Example of running the program

-> Running the program with the CC inserted and having the CC's Authentication Certificate saved on the server's folder named 'CCs'.

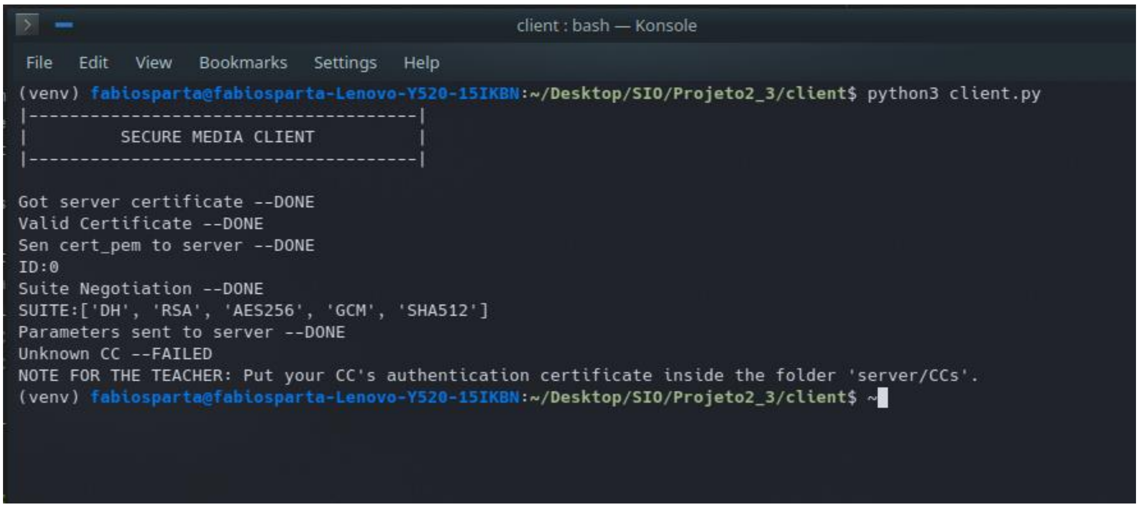


```
(venv) fabiosparta@fabiosparta-Lenovo-Y520-15IKBN:~/Desktop/SIO/Projeto2_3/client$ python3 client.py
|-------------------------------------|
|          SECURE MEDIA CLIENT        |
|-------------------------------------|

Got server certificate --DONE
Valid Certificate --DONE
Sen cert_pem to server --DONE
ID:0
Suite Negotiation --DONE
SUITE:['DH', 'RSA', 'AES128', 'GCM', 'SHA256']
Parameters sent to server --DONE
Sent CC certificate --DONE
Got Server List
MEDIA CATALOG

0 - Sunny Afternoon - Upbeat Ukulele Background Music
----
1 - 7DS - PerfectTime
----
2 - Domestic na Kanojo Opening Cover by Raon Lee
----
Select a media file number (q to quit):
```



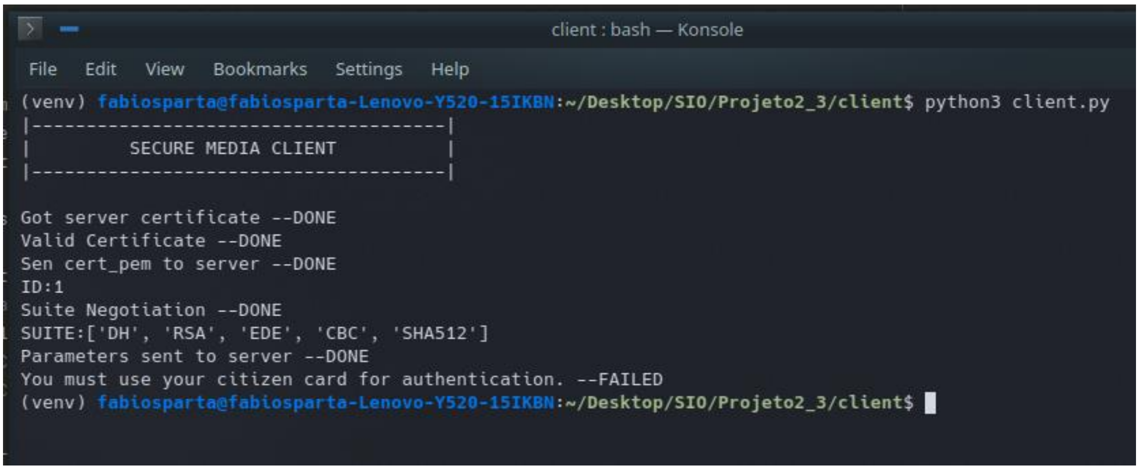
```
----
2 - Domestic na Kanojo Opening Cover by Raon Lee
----
Select a media file number (q to quit): 1
License for the music: 7DS - PerfectTime available_views: 0
Playing 7DS - PerfectTime
ffplay version 4.3.1-4ubuntu1 Copyright (c) 2003-2020 the FFmpeg developers
  built with gcc 10 (Ubuntu 10.2.0-9ubuntu2)
  configuration: --prefix=/usr --extra-version=4ubuntu1 --toolchain=harder
linux-gnu --arch=amd64 --enable-gpl --disable-stripping --enable-avresampl
e-libaom --enable-libass --enable-libbluray --enable-libbs2b --enable-libc
libflite --enable-libfontconfig --enable-libfreetype --enable-libfribidi -
 --enable-libmysofa --enable-libopenjpeg --enable-libopenmpt --enable-libc
e-librubberband --enable-libshine --enable-libsnappy --enable-libsoxr --er
enable-libtwolame --enable-libvidstab --enable-libvorbis --enable-libvpx -
l2 --enable-libxvid --enable-libzmq --enable-libzvbi --enable-lv2 --enable
--enable-pocketsphinx --enable-libmfx --enable-libdc1394 --enable-libdrm -
ei0r --enable-libx264 --enable-shared
  libavutil      56. 51.100 / 56. 51.100
  libavcodec     58. 91.100 / 58. 91.100
  libavformat    58. 45.100 / 58. 45.100
  libavdevice    58. 10.100 / 58. 10.100
  libavfilter     7. 85.100 /  7. 85.100
  libavresample   4.  0.  0 /  4.  0.  0
  libswscale      5.  7.100 /  5.  7.100
  libswresample   3.  7.100 /  3.  7.100
  libpostproc    55.  7.100 / 55.  7.100
[mp3 @ 0x7f1448000bc0] invalid concatenated file detected - using bitrate for duration
Input #0, mp3, from 'pipe:':
  Metadata:
    title           : 7DS - PerfectTime
    artist          : The Seven Deadly Sins
    encoder         : Lavf56.19.100
    major_brand     : mp42
    minor_version   : 0
    compatible_brands: isommp42
    album           : 7DS OST
    TRACKTOTAL      :
  Duration: N/A, start: 0.025057, bitrate: 192 kb/s
    Stream #0:0: Audio: mp3, 44100 Hz, stereo, fltp, 192 kb/s
    Metadata:
      encoder         : Lavc56.21
  28.78 M-A:  0.000 fd=   0 aq=   27KB vq=    0KB sq=    0B f=0/0
```

-> Running the program with the CC inserted, but forgetting to add the CC's Authentication Certificate to the server.



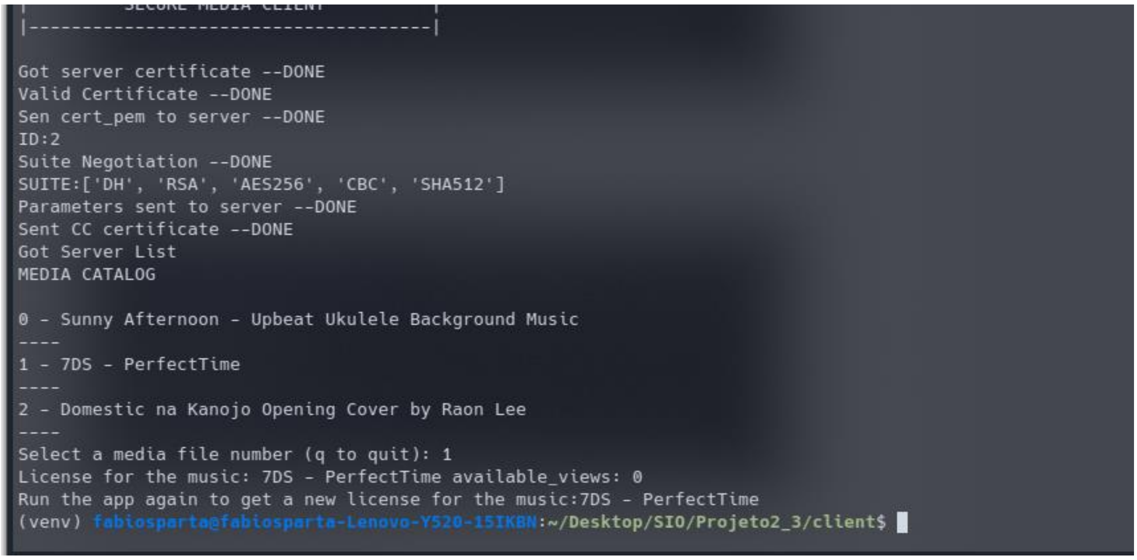
```
(venv) fabiosparta@fabiosparta-Lenovo-Y520-15IKBN:~/Desktop/SIO/Projeto2_3/client$ python3 client.py
|--------------------------------------|
|          SECURE MEDIA CLIENT         |
|--------------------------------------|

Got server certificate --DONE
Valid Certificate --DONE
Sen cert_pem to server --DONE
ID:0
Suite Negotiation --DONE
SUITE:['DH', 'RSA', 'AES256', 'GCM', 'SHA512']
Parameters sent to server --DONE
Unknown CC --FAILED
NOTE FOR THE TEACHER: Put your CC's authentication certificate inside the folder 'server/CCs'.
(venv) fabiosparta@fabiosparta-Lenovo-Y520-15IKBN:~/Desktop/SIO/Projeto2_3/client$ ~
```

-> Running the program without CC inserted.



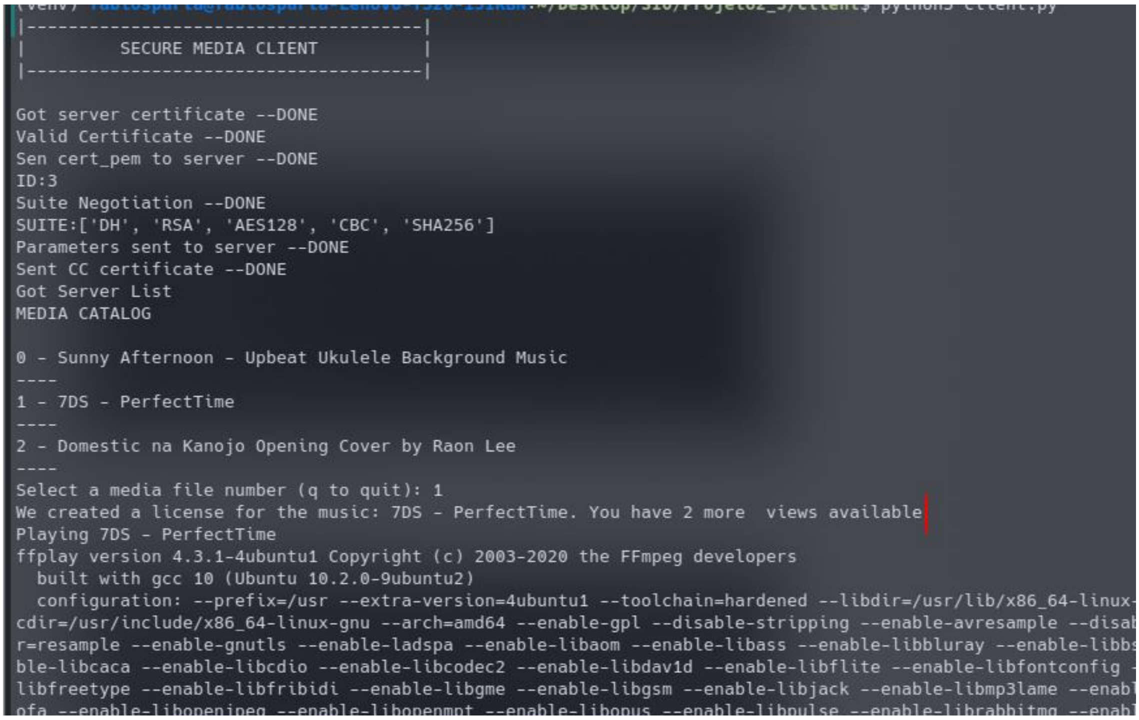
```
(venv) fabiosparta@fabiosparta-Lenovo-Y520-15IKBN:~/Desktop/SIO/Projeto2_3/client$ python3 client.py
|--------------------------------------|
|          SECURE MEDIA CLIENT         |
|--------------------------------------|

Got server certificate --DONE
Valid Certificate --DONE
Sen cert_pem to server --DONE
ID:1
Suite Negotiation --DONE
SUITE:['DH', 'RSA', 'EDE', 'CBC', 'SHA512']
Parameters sent to server --DONE
You must use your citizen card for authentication. --FAILED
(venv) fabiosparta@fabiosparta-Lenovo-Y520-15IKBN:~/Desktop/SIO/Projeto2_3/client$
```

-> Exhaust license’s available views



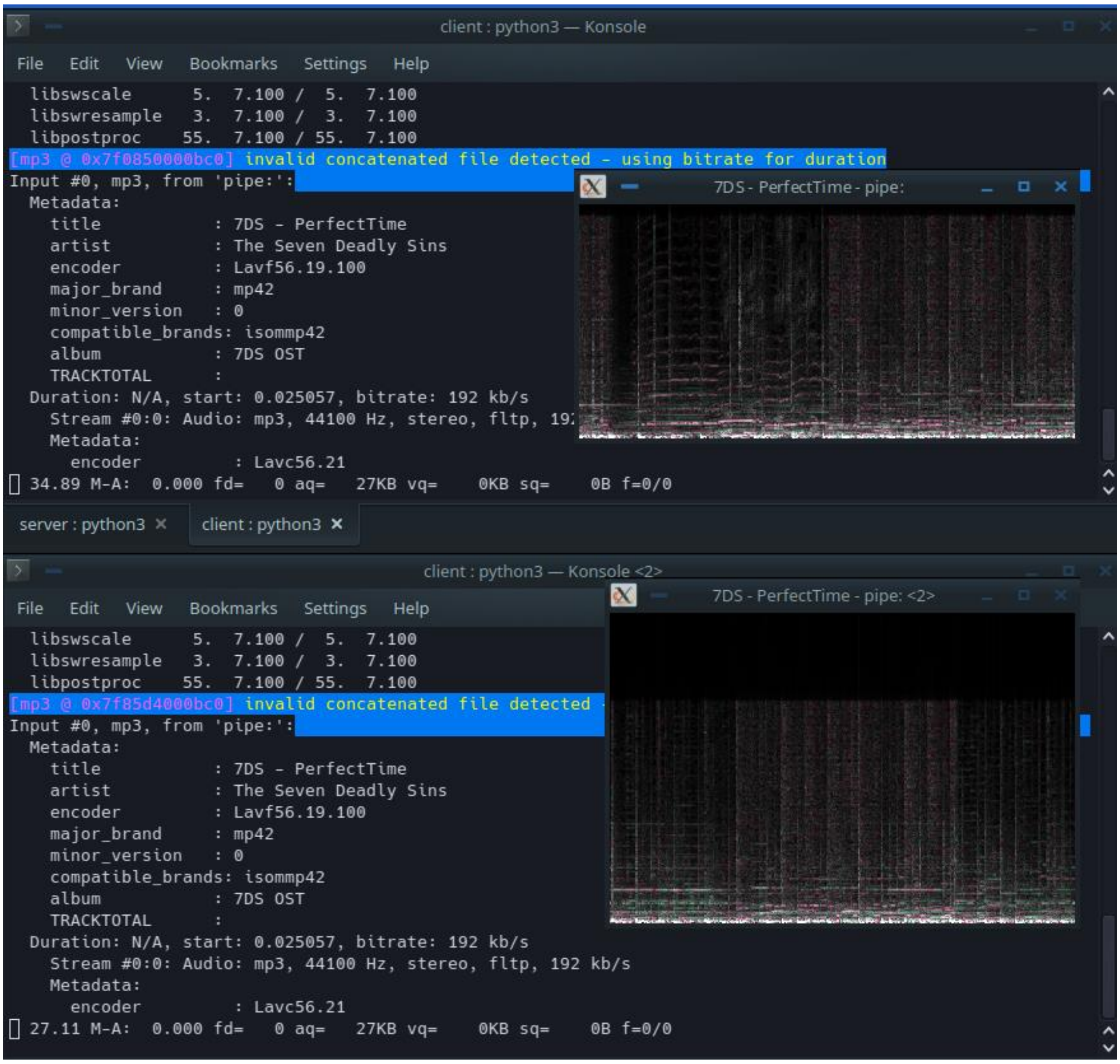
```
|          SECURE MEDIA CLIENT           |
|----------------------------------------|

Got server certificate --DONE
Valid Certificate --DONE
Sen cert_pem to server --DONE
ID:2
Suite Negotiation --DONE
SUITE:['DH', 'RSA', 'AES256', 'CBC', 'SHA512']
Parameters sent to server --DONE
Sent CC certificate --DONE
Got Server List
MEDIA CATALOG

0 - Sunny Afternoon - Upbeat Ukulele Background Music
----
1 - 7DS - PerfectTime
----
2 - Domestic na Kanojo Opening Cover by Raon Lee
----
Select a media file number (q to quit): 1
License for the music: 7DS - PerfectTime available_views: 0
Run the app again to get a new license for the music:7DS - PerfectTime
(venv) fabiosparta@fabiosparta-Lenovo-Y520-15IKBN:~/Desktop/SIO/Projeto2_3/client$
```

-> The next time the client is run, a new license is generated



```
(venv) fabiosparta@fabiosparta-Lenovo-Y520-15IKBN:~/Desktop/SIO/Projeto2_3/client$ python3 client.py
|----------------------------------------|
|          SECURE MEDIA CLIENT           |
|----------------------------------------|

Got server certificate --DONE
Valid Certificate --DONE
Sen cert_pem to server --DONE
ID:3
Suite Negotiation --DONE
SUITE:['DH', 'RSA', 'AES128', 'CBC', 'SHA256']
Parameters sent to server --DONE
Sent CC certificate --DONE
Got Server List
MEDIA CATALOG

0 - Sunny Afternoon - Upbeat Ukulele Background Music
----
1 - 7DS - PerfectTime
----
2 - Domestic na Kanojo Opening Cover by Raon Lee
----
Select a media file number (q to quit): 1
We created a license for the music: 7DS - PerfectTime. You have 2 more  views available
Playing 7DS - PerfectTime
ffplay version 4.3.1-4ubuntu1 Copyright (c) 2003-2020 the FFmpeg developers
  built with gcc 10 (Ubuntu 10.2.0-9ubuntu2)
  configuration: --prefix=/usr --extra-version=4ubuntu1 --toolchain=hardened --libdir=/usr/lib/x86_64-linux-
cdir=/usr/include/x86_64-linux-gnu --arch=amd64 --enable-gpl --disable-stripping --enable-avresample --disab
r=resample --enable-gnutls --enable-ladspa --enable-libaom --enable-libass --enable-libbluray --enable-libbs
ble-libcaca --enable-libcdio --enable-libcodec2 --enable-libdav1d --enable-libflite --enable-libfontconfig -
libfreetype --enable-libfribidi --enable-libgme --enable-libgsm --enable-libjack --enable-libmp3lame --enabl
ofa --enable-libopenjpeg --enable-libopenmpt --enable-libopus --enable-libpulse --enable-librabbitmq --enabl
```

-> Running multiple clients at the same time


client : python3 — Konsole
File Edit View Bookmarks Settings Help
7DS - PerfectTime - pipe:
server : python3
client : python3
client : python3 — Konsole <2>
7DS - PerfectTime - pipe: <2>

```
  libswscale      5.  7.100 /  5.  7.100
  libswresample   3.  7.100 /  3.  7.100
  libpostproc    55.  7.100 / 55.  7.100
[mp3 @ 0x7f0850000bc0] invalid concatenated file detected - using bitrate for duration
Input #0, mp3, from 'pipe:':
  Metadata:
    title           : 7DS - PerfectTime
    artist          : The Seven Deadly Sins
    encoder         : Lavf56.19.100
    major_brand     : mp42
    minor_version   : 0
    compatible_brands: isommp42
    album           : 7DS OST
    TRACKTOTAL      :
  Duration: N/A, start: 0.025057, bitrate: 192 kb/s
    Stream #0:0: Audio: mp3, 44100 Hz, stereo, fltp, 19
    Metadata:
      encoder         : Lavc56.21
  34.89 M-A:  0.000 fd=   0 aq=   27KB vq=    0KB sq=    0B f=0/0
```

```
  libswscale      5.  7.100 /  5.  7.100
  libswresample   3.  7.100 /  3.  7.100
  libpostproc    55.  7.100 / 55.  7.100
[mp3 @ 0x7f85d4000bc0] invalid concatenated file detected -
Input #0, mp3, from 'pipe:':
  Metadata:
    title           : 7DS - PerfectTime
    artist          : The Seven Deadly Sins
    encoder         : Lavf56.19.100
    major_brand     : mp42
    minor_version   : 0
    compatible_brands: isommp42
    album           : 7DS OST
    TRACKTOTAL      :
  Duration: N/A, start: 0.025057, bitrate: 192 kb/s
    Stream #0:0: Audio: mp3, 44100 Hz, stereo, fltp, 192 kb/s
    Metadata:
      encoder         : Lavc56.21
  27.11 M-A:  0.000 fd=   0 aq=   27KB vq=    0KB sq=    0B f=0/0
```

Conclusion

Through the development of this project, we were able to achieve all the stipulated goals while putting to practice what was learnt during classes, deepening our understanding of how to better implement secure communication channels.

Bibliography and References

- cryptography.io
- StackOverflow
- https://joao.barraca.pt/teaching/sio/2020/