Digital Rights Management

Universidade de Aveiro

Licenciatura em Engenharia Informática

UC 42573 - Segurança Informática e nas Organizações

Docentes:

Prof. João Paulo Barraca
Prof. Vitor Cunha

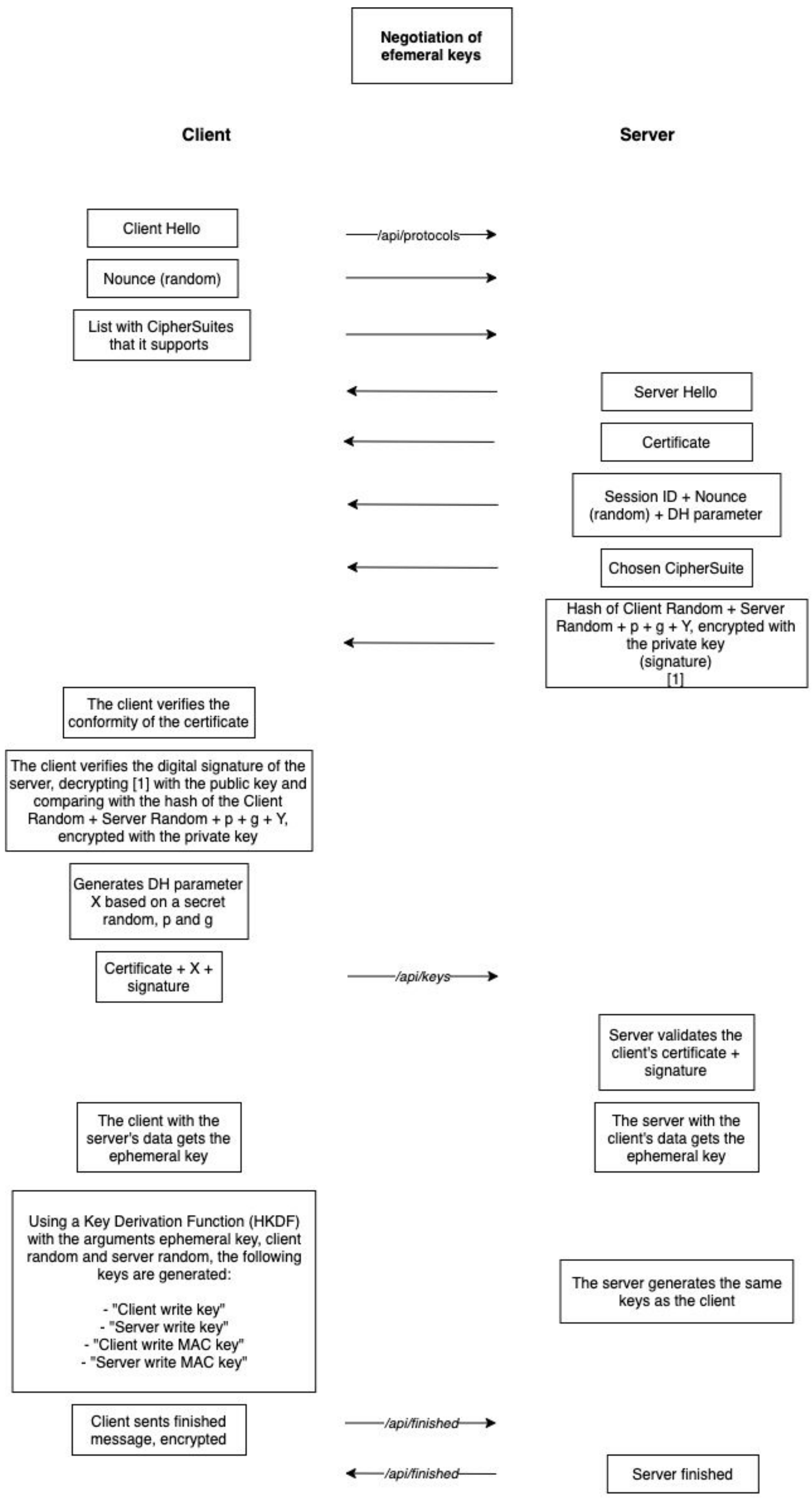
Trabalho realizado por:

Eduardo Santos - 93107
Margarida Martins - 93169

Index

1 - Protocol Diagram

Negotiation of Ephemeral Keys Diagram



Negotiation of Ephemeral Keys Step By Step

Step 1 - Client Hello

In order to connect with the server the client sends a post request with the url path ***/api/protocols.***
In the request data the client sends a ***client_random*** (random string of 28 bytes) and a list of **cipher suites** he supports.

Step 2 - Server Hello

When the server receives a ***/api/protocols*** post request it will choose a **cipher suite** based on its preferences and the cipher suites supported by the client. In the project context the server chooses a cipher suite, that both he and the client support, randomly.
The server will also generate a random string of 28 bytes, the ***server_random***.
In order to do the ephemeral diffie hellman key exchange the server will generate a modulus **p** and a base **g.** With these parameters and a secret random he will calculate a **Y.** These calculations are done in the ***generate_DH_parameter*** function.
The server will send to the client his certificate, the ***server_random***, **p**, **g, Y** and, in order to prove its authenticity, a signature i.e. an hash of the concatenation of the ***client_random, server_random,*** **p**, **g** and **Y** encrypted with his private key, the signature is made using the function ***make_signature***.
In the end the server stores the chosen **cipher suite**, the ***client_random,*** the ***server_random*** and the ***DH parameters*** in the ***SESSIONS*** dictionary where the key is the session created for the client.

Step 3 - Client Certificate and DH parameters

The client validates the server certificate using the ***verify_server_certificate*** function. The function checks the certificate validation dates (not_valid_before and not_valid_after) the subject and issuer ***COMMON_NAME,*** some key usages and verifies the certificate signature using the ***ROOT_CA*** public key.
In order to validate the signature sent the client will use the ***server certificate*** public key and the concatenation of the ***client_random, server_random,*** **p**, **g** and **Y** as the signed data.
After validating the server's authenticity. The client will generate its ***DH parameter*** (**X**) based on a secret random, **p** and **g**.
Using the same process as the server he will create a signature using as info the ***client_random,*** the ***server_random*** and **X**.
In the end the client will send a post request with the url path ***/api/keys*** with the following data: ***client certificate***, ***X*** and the created signature.

Step 4 - Server client authenticity verification

When the server receives a ***/api/keys*** post request it will first check in the ***SESSIONS*** dictionary if the previous steps were made (see if there is a key equal to the request session in the dictionary). If that client did not do the first steps the server returns an error message.

Using the same process as the client in the Step 3 the server will validate the clients authenticity (certificate and signature validation).
It will also store **X** which will be needed to generate the sessions key.

Step 5 - Session Keys Generation

At this point both the client and the server have all that is needed to generate the sessions keys. Using the **p**, **g**, the secret random and **Y** or **X** (**Y** for the client and **X** for the server) client and server calculate a secret key the key should be the same.
Using a Hash Key Derivation Function, with the salt being the concatenation of the ***client_random*** and the ***server_random,*** they generate the following session keys:

- ***client_write_key***
  - Will be used to encrypt and decrypt client messages
- ***client_write_mac_key***
  - Will be used to generate MACs in the client messages in order to validate the authenticity and integrity of a message
- ***server_write_key***
  - Will be used to encrypt and decrypt server messages
- ***server_write_mac_key***
  - Will be used to generate MACs in the server messages in order to validate the authenticity and integrity of a message

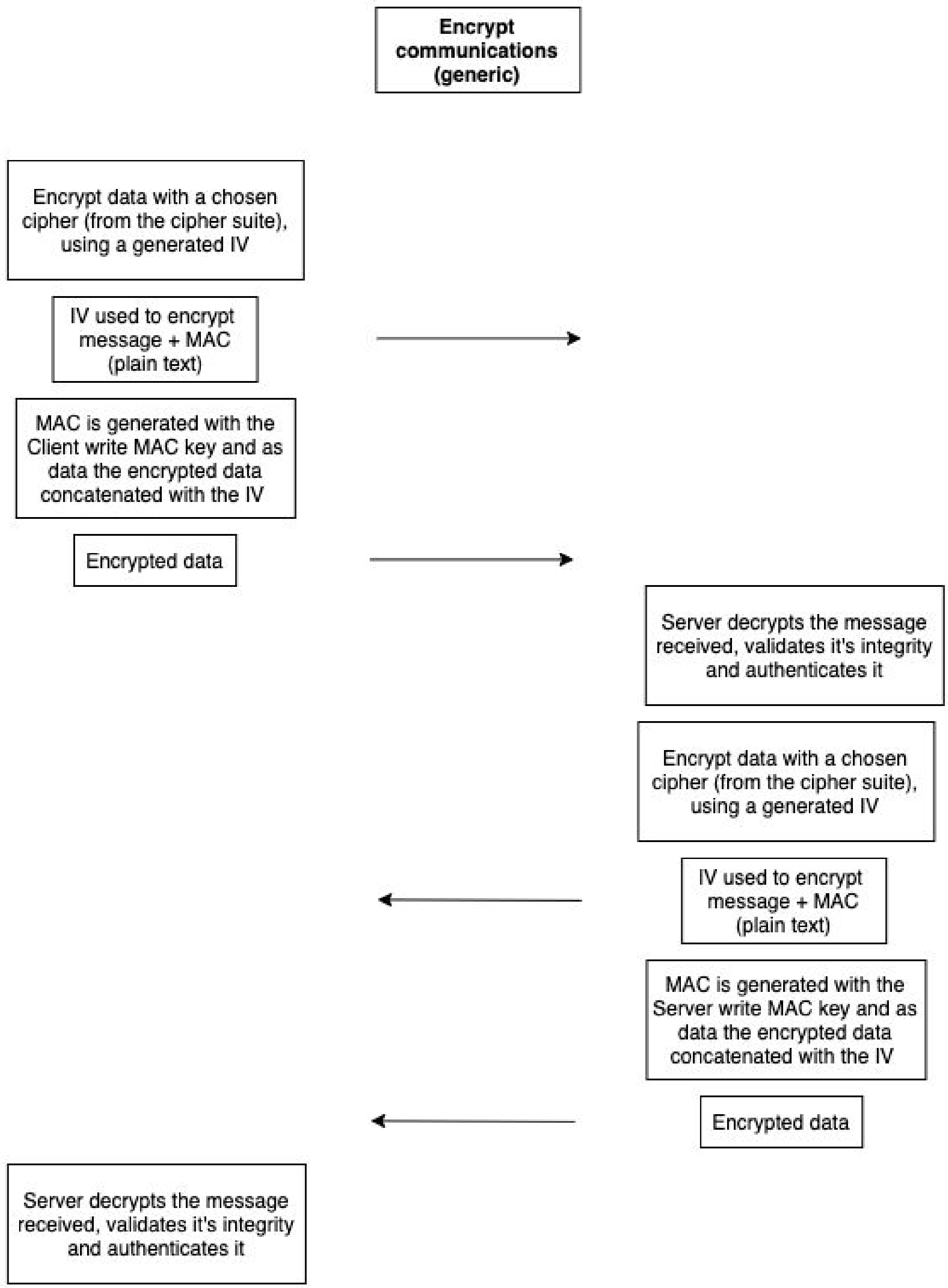
Step 6 - Client Finished

After Step 5 the client will send an encrypted get request with the url path ***api/finished.***
The encryption is done using the client keys generated in Step 5, the detailed encryption method is explained in the *Encrypt communications (generic)* part of the report.

Step 7 - Server Finished

When the server is able to decrypt a request with the path ***api/finished*** it sets the client in the ***SESSIONS*** dictionary as finished. He will then respond with a “finished” message encrypted with the server keys generated in Step 5. The message will be decrypted by the client.
In this state both the client and server have verified each other keys and are ready to communicate more securely. From now on **ALL** communications will be encrypted.

Encrypt and Decrypt Communications (generic) Diagram



Encrypt and Decrypt communications (generic) Step by Step

Step 1 - Client encrypts and sends message

In every encrypted message sent by the client only the server base url (in this case ***http://127.0.0.1:8080***), the **IV** used to encrypt and the **MAC** are sent in plain text. Everything else will be encrypted, the url path and the data sent.

In order to encrypt the data the client uses the function ***encrypt_comunication.*** This function will generate an **IV** and encrypt the data with the correct cipher depending on the chosen **cipher suite.** Depending on the cipher modes the data might need padding, the padding is done using ***padding_data*** function which itself uses ***PKCS7***.
Along with the encryption a MAC is generated using the function ***generate_hmac*** in this process is used as key the ***client_write_mac_key*** and as data the encrypted data concatenated with the **IV**.

Step 2 - Server decrypts message

The server uses de function ***decrypt_comunication*** in order to decrypt the messages sent by the client, validate their integrity and authenticate them.
First a MAC is generated with the encrypted_data concatenated with the **IV** using the ***client_write_mac_key.*** If this MAC is equal to the MAC sent by the client the data is validated, else the server returns an error message.
After validating the encrypted data the server will decrypt it using the function ***decrypt_symetric*** where the used key will be the ***client_write_key.*** Depending on the cipher modes the decrypted data might need to be unpadded, to unpad the server uses the ***unpadding_data*** function.

Step 3 - Server encrypts a message

When responding to a request the server does the same encrypting process as the client (Step 1) using its keys (***server_write_key*** and ***server_write_mac_key***).
The server sends a json with a field data which has the **IV, MAC,** and encrypted data concatenated.

Step 4 - Client decrypts a message

When the client receives an encrypted response from the server it will get the 'data' key in the response json. It will then use the same decrypting process as the server (Step 2) with the server keys (***server_write_key*** and ***server_write_mac_key***).

Authenticate User Diagram

Authenticate User Step by Step

Step 1 - Client: Getting and sending user token information

The Portuguese Citizen Card was the token chosen to authenticate an user.
In order to access the token the ***libpteidpkcs11.so*** library was used.
First, in the client ***user_authentication*** function using the **PyKCS11 Session** method ***findObjects*** we are able to fetch the users private key ("CITIZEN AUTHENTICATION KEY"), the user certificate ("CITIZEN AUTHENTICATION CERTIFICATE"), the intermediate certificate ("AUTHENTICATION SUB CA"), and the root certificate ("ROOT CA").
The three certificates form the certificate chain that will be sent to the server. With the user's private key and the **PyKCS11 Session** method ***sign*** a signature is made with the user's certificate serial number (which corresponds to the user civil ID) as data.
Both the signature and the certificate chain are sent to the server using the encrypted path ***api/auth.***

Step 2 - Server: Verifying Signature

The server receives the certificate chain and the signature from the client.
He will load the user certificate, get its public key and its serial number. With this data he will check the validity of the signature using the function ***user_verify_signature.*** In this function differs from the ***verify_signature,*** because the padding function needs to be ***padding.PKCS1v15(),*** and the hash algorithm ***SHA1.*** This is due to how the signature was made, as explained in Step 1.

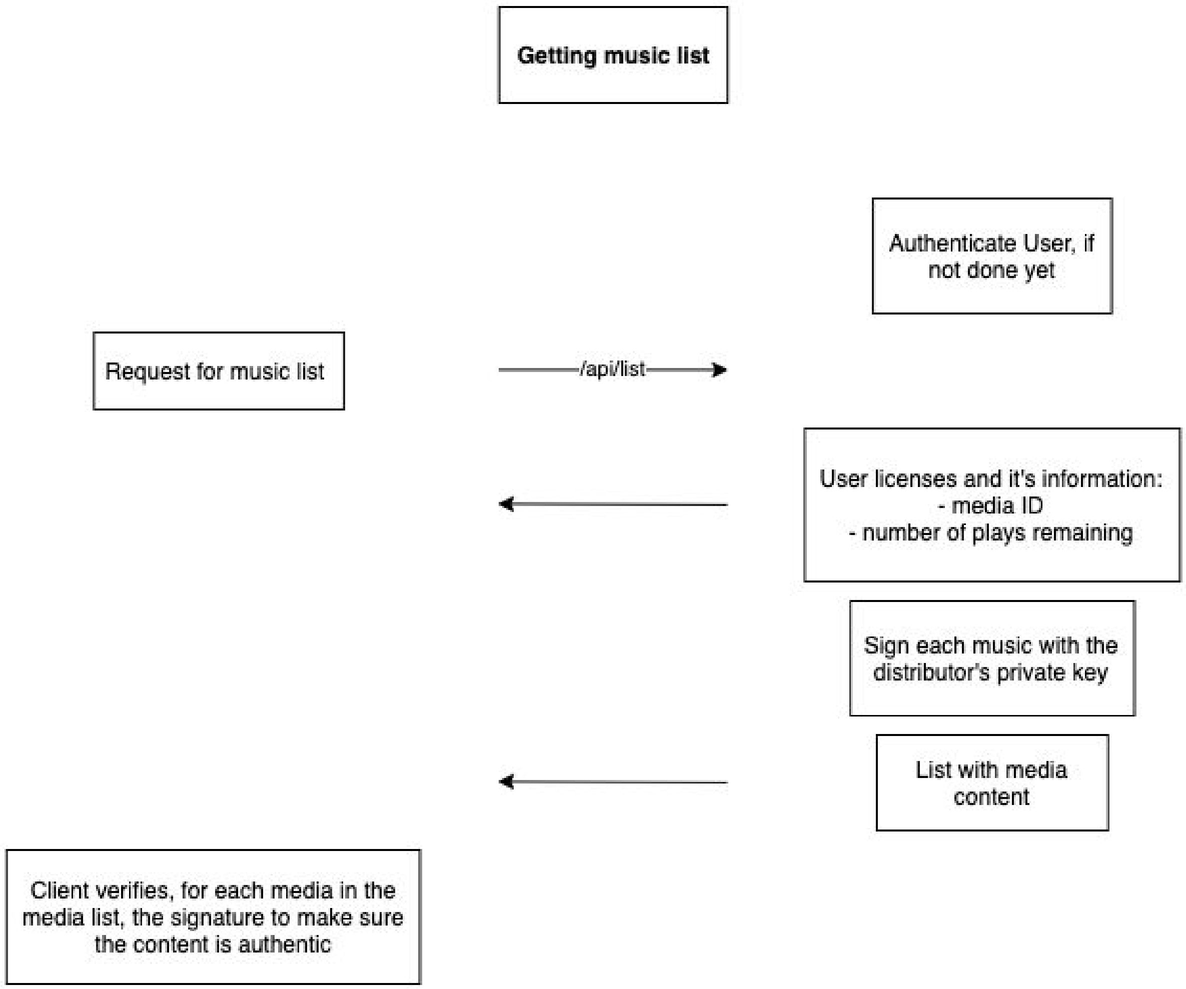
Step 3 - Server: Validation Certificate Chain

The validation of the certificate chain was divided in three sub-steps:

- Validating the certificates' attributes (***validate_attributes*** function)
    - Validation of the organization name
    - Validation of the certificate validation dates (not_valid_before and not_valid_after)
    - Validation of the certificate key_usages
- Check for the certificate in the CRL (***validate_crl*** function)
    - For the user certificate and the intermediate one, we download the crl and delta crl lists (if existent) from the endpoint in the ***CRL_DISTRIBUTION_POINTS*** and ***FRESHEST_CRL***
    - The lists are ***loaded x509.load_der_x509_crl*** and its verified if the certificate is in the CRL the certificate serial number using the function ***get_revoked_certificate_by_serial_number***
- Validate the certificates' signatures
    - For the user certificate and the intermediate one, we verify the certificate ***signature*** filed using the issuer public key and the fields ***tbs_certificate_bytes*** and ***signature_hash_algorithm***

After validating the certification chain the user is authenticated. In the ***SESSIONS*** dictionary is stored the user civil ID.

Getting Music List Diagram



Getting Music List Set by Step

Step 1 - Client: Request for Media and License List

In order to request for media and license list the client sends to the server a get request with the encrypted path ***api/list.***

Step 2 - Server: Get all licenses from users

When receiving a media and license list request the server checks if the user has authenticated by checking if the ***USER_ID*** key exists in the ***SESSIONS*** dictionary. If the user isn't authenticated the server returns an error message.
Using the client certificate ***COMMON_NAME*** and the ***USER_ID*** hashed we can get all user licenses from the licenses folder.

For each license we append the media id and the number of plays to the license list which is going to be sent to the client.

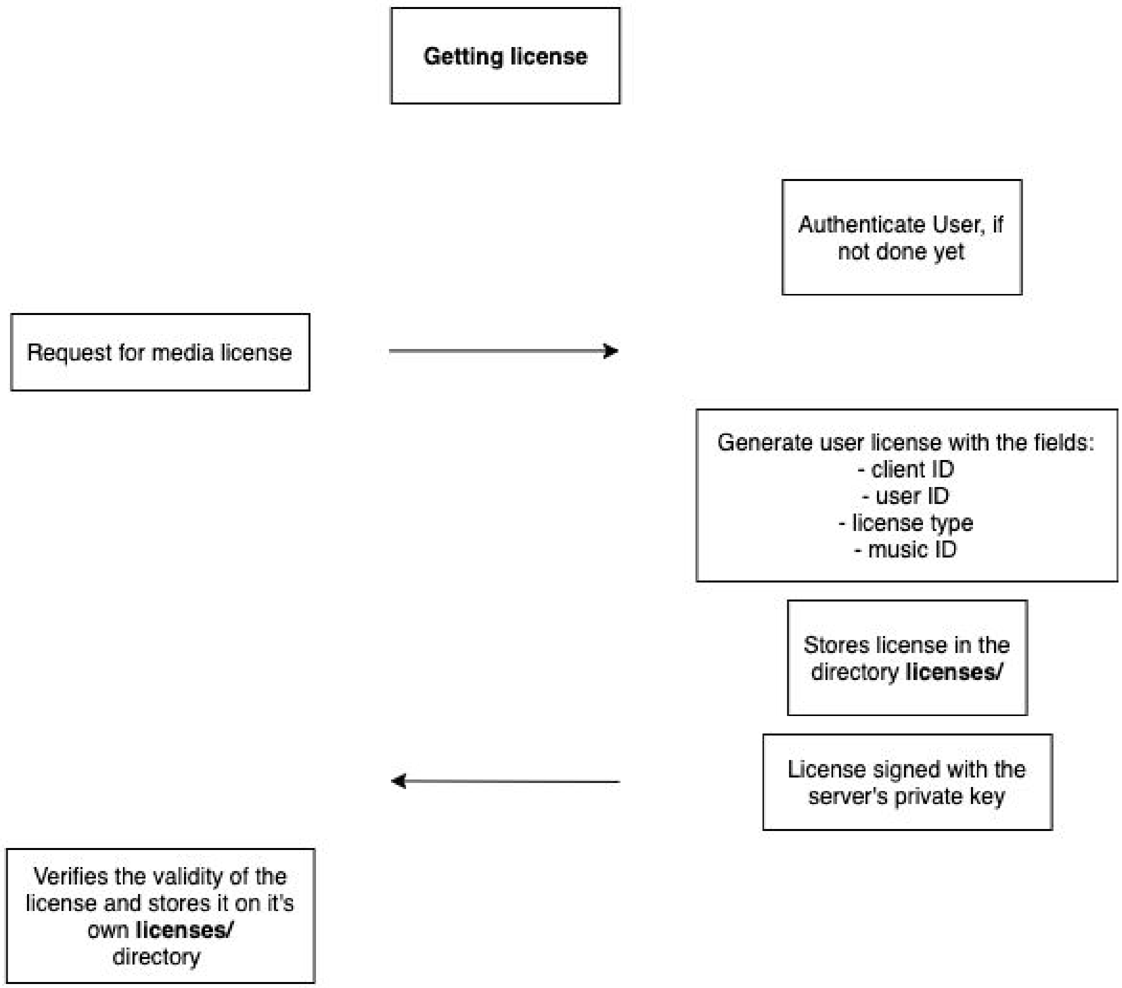
Step 3 - Server: Get media list

For each media in the catalog it is appended to the media list the media info and a distributor signature. Each music is signed with its distributor private key, for the context of the project we considered only one distributor for all the media.
The server then returns an encrypted json containing both lists.

Step 4 - Validating the lists

The client, for each media in the media list, using the distributor public key will verify the signature making sure the content is authentic.
For each license sent by the server the client will corroborate its information i.e. seeing if the server didn’t cheat.

Getting license



Getting license Step by Step

Step 1 - Client: Request for media license

The user requests to acquire a license, and then chooses a license type, each one containing a different number of granted plays (1, 5, 10 or 20).
After the number of plays being specified, the user has to choose the license media.
Then, the license is sended to the server through a get request to the encrypted path **api/license.**

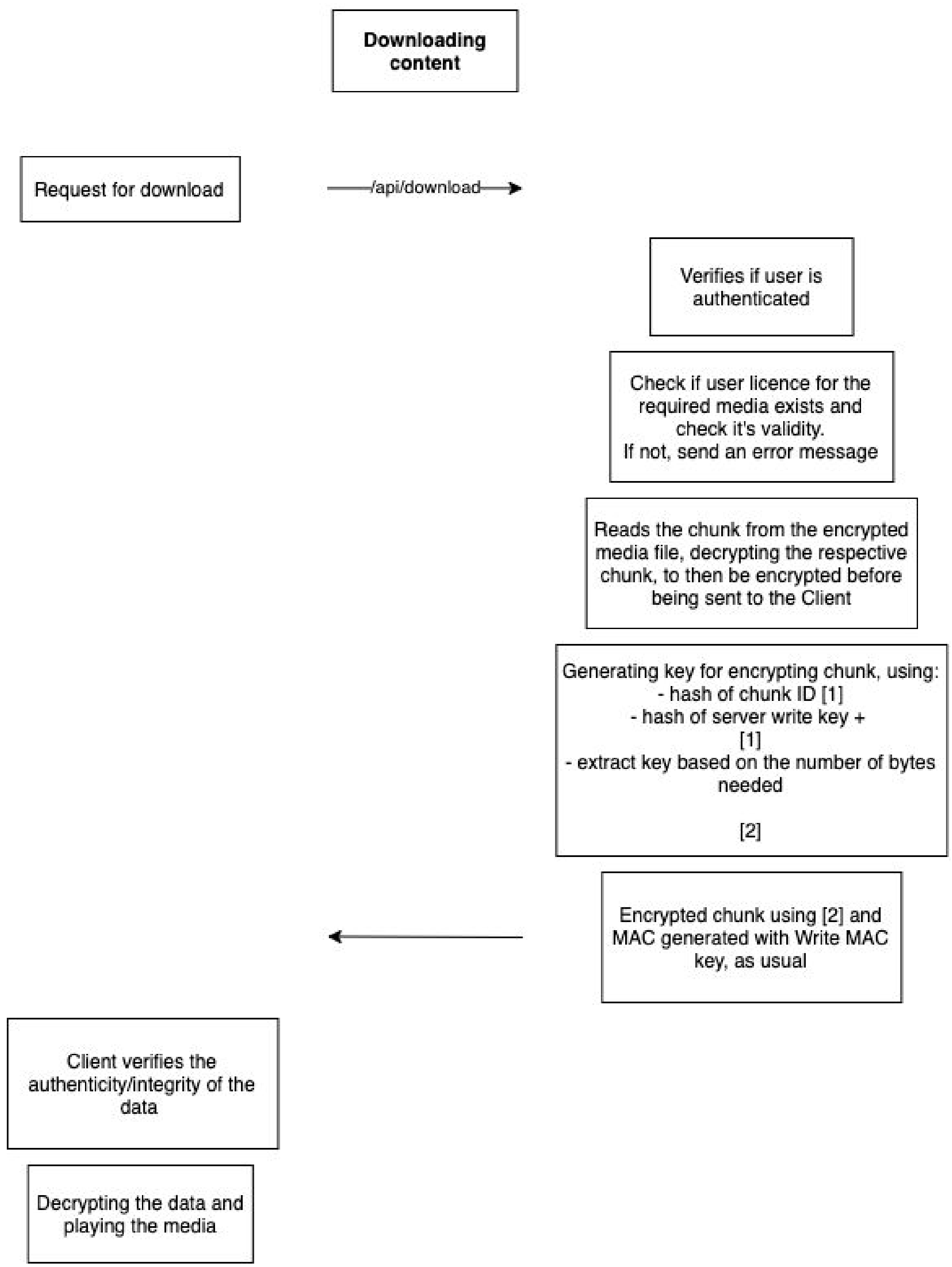
Step 2 - Server: Generating user license

The server receives the request with the license, and then checks if the user is authenticated, if so, a license is generated with the required fields (client_id, user_id, license_type and music_id).
When that is done, the license is kept by the server stored in the **licenses/** directory. Then the server will then send the license to the client signed with his private key.

Step 3 - Client: Receiving user license

When the client receives the license he will first check the validity of the signature using the servers public key and then store the license in the **licenses/** directory.

Downloading Content Diagram



Downloading Content Step by Step

Step 1 - Client: Request for Download

In order to play a media the client will continuously request the server for chunks of that media until the last chunk. The client can know how many chunks the music has by doing a media list request.
The client will check within its records if the user has a license to the music.
The get request is made using the encrypted path ***api/download***.

Step 2 - Server: Verifying a User

When receiving an ***api/download*** request the server wil check in the ***SESSIONS*** dictionary if the user is authenticated (returns an error message if not).
If the chunk is the first one (id=0), the server will fetch the user license for that media with that client. If the user does not have a valid license the server returns an error message.

Step 3 - Server: Decrypting Chunk

The media content at rest in the server is encrypted using AES in ECB mode.
So in order to send the chunk to the client the server first reads the chunk from the respective media file and then decrypts it using the function ***decrypt_data.*** This function decrypts the data using as key, a key derived from the ***FILE_DECRYPTION_KEY*** using as salt the ***FILE_DECRYPTION_SALT.*** It then checks through the variable ***padding_flag*** whether the data will need padding i.e is the last file chunk.

Step 4 - Server: Encrypting and Sending Chunk

In order to increase the security the chunk response is encrypted using a chunk based key rotation.
First the server hashes (using the chosen cipher suite digest) the chunk id. Then the server hashes the concatenation of the ***server_write_key*** with the previous hash. With the result the key to the encryption is extracted (the first 16 or 32 bytes depending on the cipher).
Using this key the data is encrypted and sent (like the rest of the communications encrypted the mac is generated using the ***server_write_mac_key***).

Step 5 - Client: Decrypting Sent Chunk

Like always, the client first checks the authenticity/integrity of the data received using the MAC sent and the ***server_write_mac_key***.
Then in order to decrypt the response data sent by the server the client must get the same key the server used to encrypt. So it uses the same process as the user to generate the key. First hashes the chunk id, then hashes the result concatenated with the ***server_write_key*** and extracts the final key.
This way the keys used to encrypt one chunk and the next one will be totally different.

2 - Cipher Suites and Certificates

Cipher Suites

We used 7 types of cipher suites, containing, at least, 2 ciphers, 2 digests and 2 cipher modes. This cipher suites being:

1. DHE_AES256_CBC_SHA384
2. DHE_AES256_CFB_SHA384
3. DHE_AES128_CBC_SHA256
4. DHE_AES128_CBC_SHA384
5. DHE_ChaCha20_SHA384
6. DHE_ChaCha20_SHA384
7. DHE_ChaCha20_SHA256

The **Key Exchange Algorithm** used was **Diffie-Hellman Ephemeral**, we also have 3 **Bulk Encryption Algorithms** (**AES128, AES256** and **ChaCha20**), with 2 different modes (**CBC** and **CFB**) and 2 different **MAC Algorithms** (**SHA256** and **SHA384**).

Certificates

Using xca we created a database mediaDB.xdb
Using that database we created a certificate for the ***ROOT_CA*** (***Media_CA.crt).***
Three more certificates were created all signed by the ***ROOT_CA***:

- Media Distributor Certificate (***Media_Distributor.crt***)
    - Media Distributor Private Key with password (***/server/Media_Distributor_Private_Key.pem***)
- Media Server Certificate (***Media_Server.crt***)
    - Media Server Private Key with password (***/server/Media_Server_Private_Key.pem***)
- Media Client Certificate (***Media_Client.crt***)
    - Media Client Private Key with password (***/client/Media_Client_Private_Key.pem)***

3 - Operation of the features implemented

Client Side

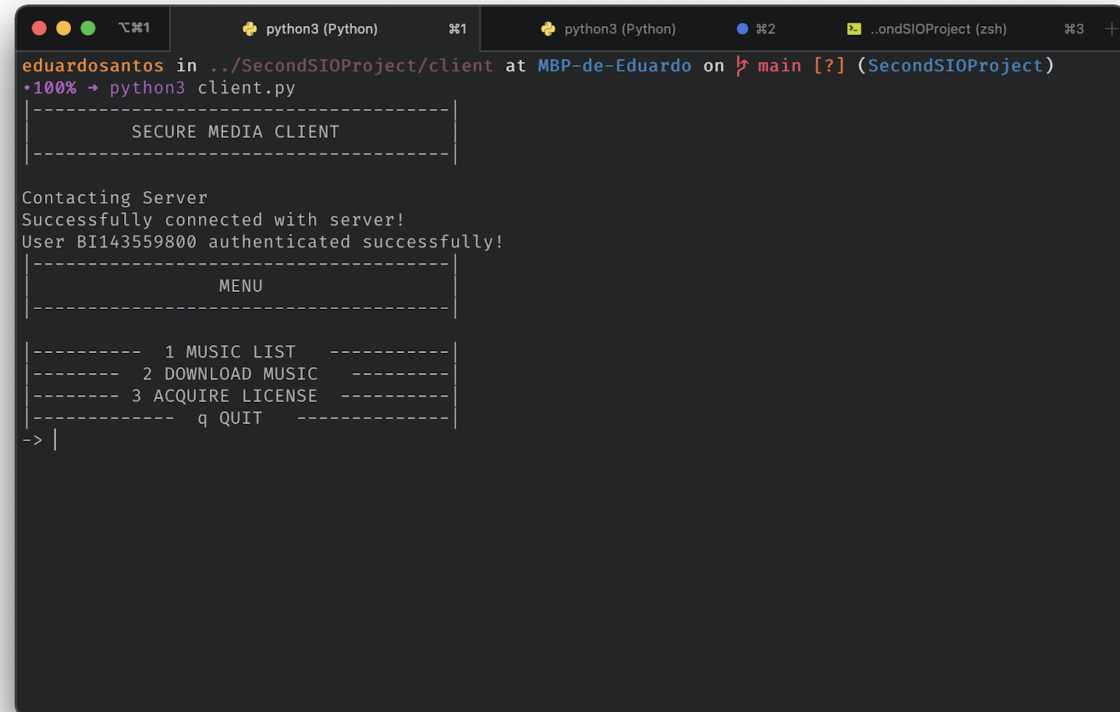
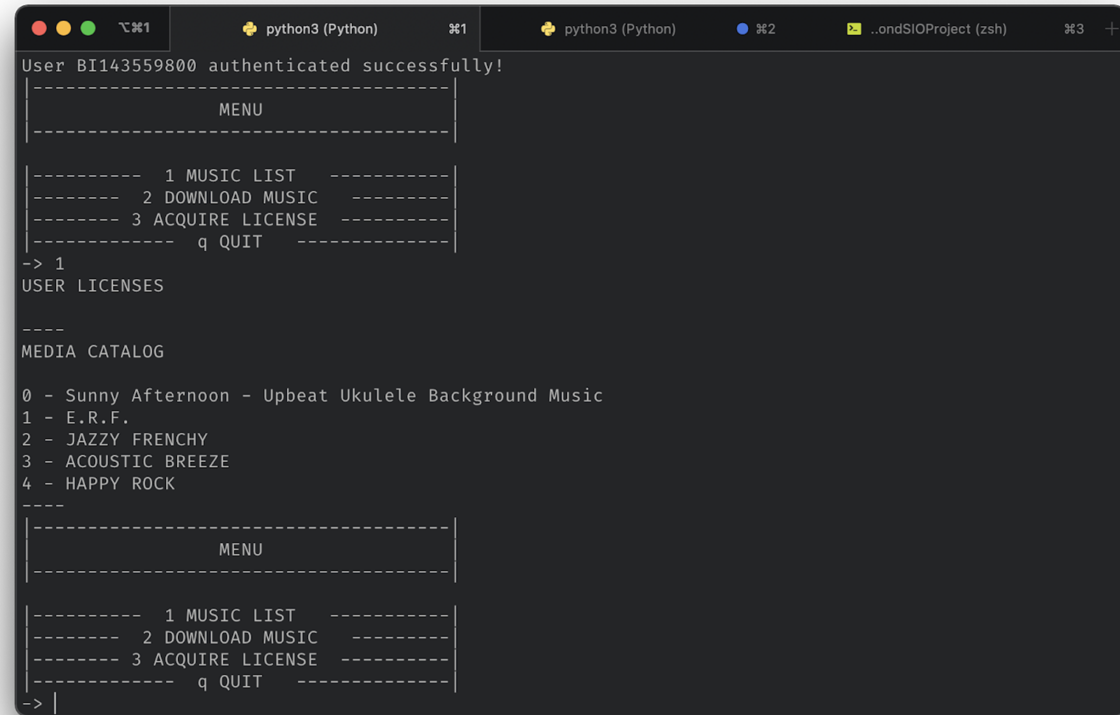


fig 1. Initial menu

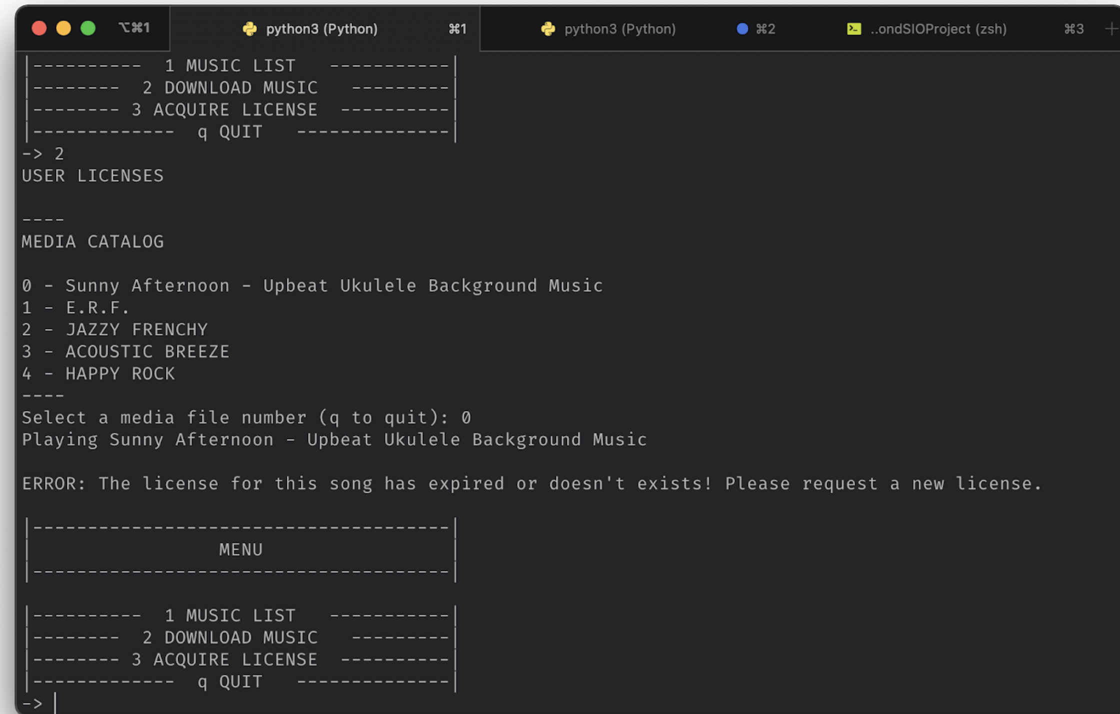
The user can see the music and license list by choosing option 1, play music by choosing option 2, acquire a license by choosing option 3 and quitting the program by choosing option q. When the user chooses the quit option the client will send an api/exit request to the server. The server will then delete the respective session data in the SESSIONS dictionary.




```
User BI143559800 authenticated successfully!
|------------------------------------|
|                MENU                |
|------------------------------------|

|----------   1 MUSIC LIST   -----------|
|--------  2 DOWNLOAD MUSIC   ---------|
|-------- 3 ACQUIRE LICENSE  ----------|
|-------------   q QUIT   -------------|
-> 1
USER LICENSES

----
MEDIA CATALOG

0 - Sunny Afternoon - Upbeat Ukulele Background Music
1 - E.R.F.
2 - JAZZY FRENCHY
3 - ACOUSTIC BREEZE
4 - HAPPY ROCK
----
|------------------------------------|
|                MENU                |
|------------------------------------|

|----------   1 MUSIC LIST   -----------|
|--------  2 DOWNLOAD MUSIC   ---------|
|-------- 3 ACQUIRE LICENSE  ----------|
|-------------   q QUIT   -------------|
->
```

fig 2. Listing media catalog

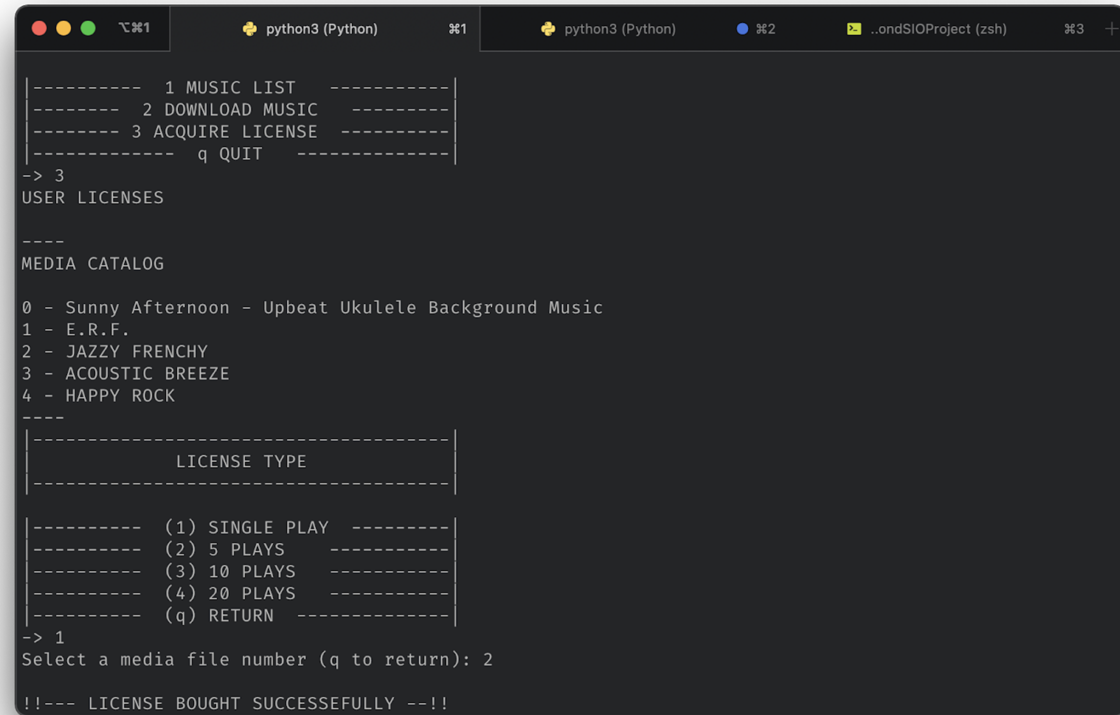
When the user chooses option 1 a list with all user licenses (for that client) is shown (name of music and number of plays). A list with the name of all available songs is printed.




```
|----------  1 MUSIC LIST   -----------|
|--------  2 DOWNLOAD MUSIC   ---------|
|-------- 3 ACQUIRE LICENSE  ----------|
|-------------  q QUIT   --------------|
-> 2
USER LICENSES

----
MEDIA CATALOG

0 - Sunny Afternoon - Upbeat Ukulele Background Music
1 - E.R.F.
2 - JAZZY FRENCHY
3 - ACOUSTIC BREEZE
4 - HAPPY ROCK
----
Select a media file number (q to quit): 0
Playing Sunny Afternoon - Upbeat Ukulele Background Music

ERROR: The license for this song has expired or doesn't exists! Please request a new license.

|--------------------------------------|
|                 MENU                 |
|--------------------------------------|

|----------  1 MUSIC LIST   -----------|
|--------  2 DOWNLOAD MUSIC   ---------|
|-------- 3 ACQUIRE LICENSE  ----------|
|-------------  q QUIT   --------------|
-> |
```

fig 3. Choosing to play a song without any license

If the user chose to download a media that he has no license, he will not be able to play it and an error message will be displayed.

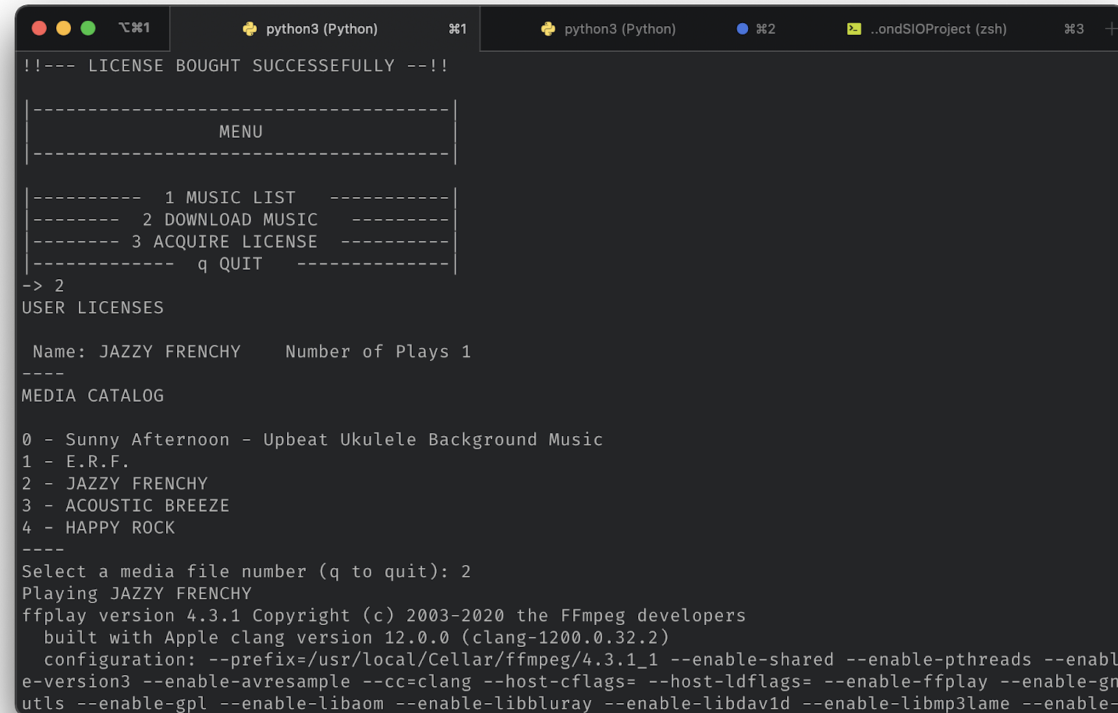


```
|----------  1 MUSIC LIST   -----------|
|--------  2 DOWNLOAD MUSIC   ---------|
|-------- 3 ACQUIRE LICENSE  ----------|
|-------------   q QUIT   -------------|
-> 3
USER LICENSES

----
MEDIA CATALOG

0 - Sunny Afternoon - Upbeat Ukulele Background Music
1 - E.R.F.
2 - JAZZY FRENCHY
3 - ACOUSTIC BREEZE
4 - HAPPY ROCK
----
|--------------------------------------|
|             LICENSE TYPE             |
|--------------------------------------|

|----------  (1) SINGLE PLAY  ---------|
|----------  (2) 5 PLAYS    -----------|
|----------  (3) 10 PLAYS   -----------|
|----------  (4) 20 PLAYS   -----------|
|----------  (q) RETURN  --------------|
-> 1
Select a media file number (q to return): 2

!!--- LICENSE BOUGHT SUCCESSEFULLY --!!
```

fig 4. Acquire license

When the user chooses the option 3 Acquire License he can choose 4 different types of license: a single play license, 5, 10 and 20 plays license.

After chosing the license type the user can introduce a number corresponding to the respective media.



```
!!--- LICENSE BOUGHT SUCCESSEFULLY --!!

|-------------------------------------|
|                MENU                 |
|-------------------------------------|

|----------  1 MUSIC LIST   -----------|
|--------  2 DOWNLOAD MUSIC   ---------|
|-------- 3 ACQUIRE LICENSE  ----------|
|------------  q QUIT   -------------|
-> 2
USER LICENSES

 Name: JAZZY FRENCHY    Number of Plays 1
----
MEDIA CATALOG

0 - Sunny Afternoon - Upbeat Ukulele Background Music
1 - E.R.F.
2 - JAZZY FRENCHY
3 - ACOUSTIC BREEZE
4 - HAPPY ROCK
----
Select a media file number (q to quit): 2
Playing JAZZY FRENCHY
ffplay version 4.3.1 Copyright (c) 2003-2020 the FFmpeg developers
  built with Apple clang version 12.0.0 (clang-1200.0.32.2)
  configuration: --prefix=/usr/local/Cellar/ffmpeg/4.3.1_1 --enable-shared --enable-pthreads --enabl
e-version3 --enable-avresample --cc=clang --host-cflags= --host-ldflags= --enable-ffplay --enable-gn
utls --enable-gpl --enable-libaom --enable-libbluray --enable-libdav1d --enable-libmp3lame --enable-
```

fig 5. Playing a song

After successfully choosing a music to download the ffplay will be initiated and the user can hear the music

Server Side

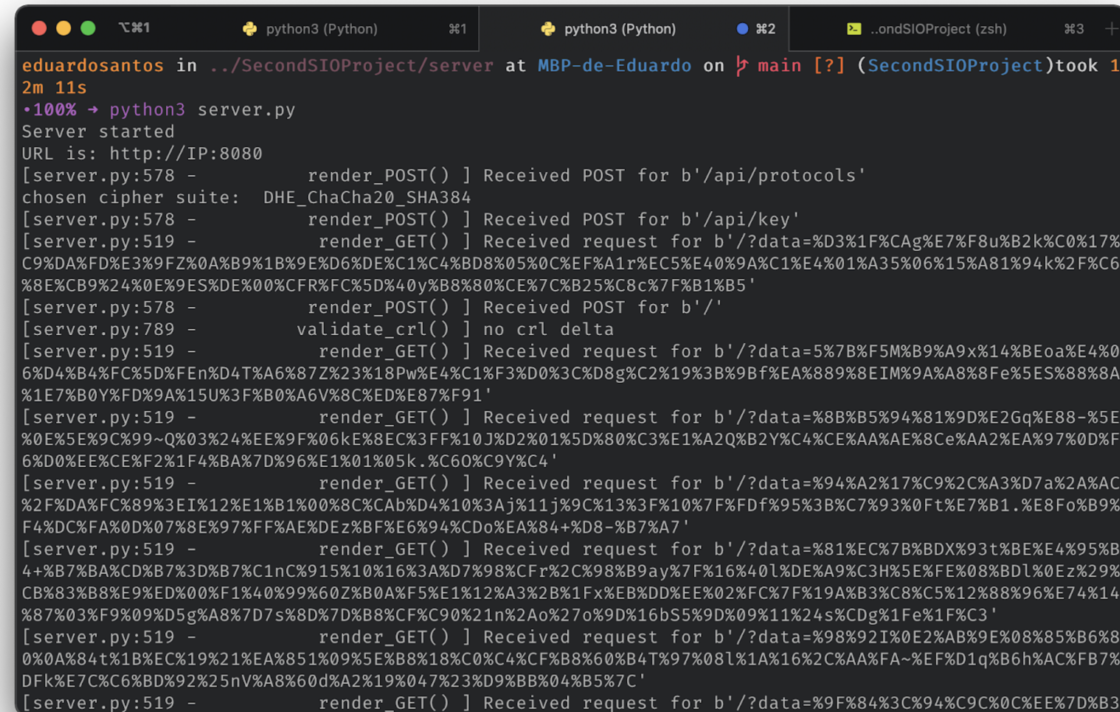


fig 6. server Negotiation of Ephemeral Keys

In this picture we can see clearly that the first two messages /api/protocols and /api/key are the only ones not encrypted. The rest of the messages look like noise

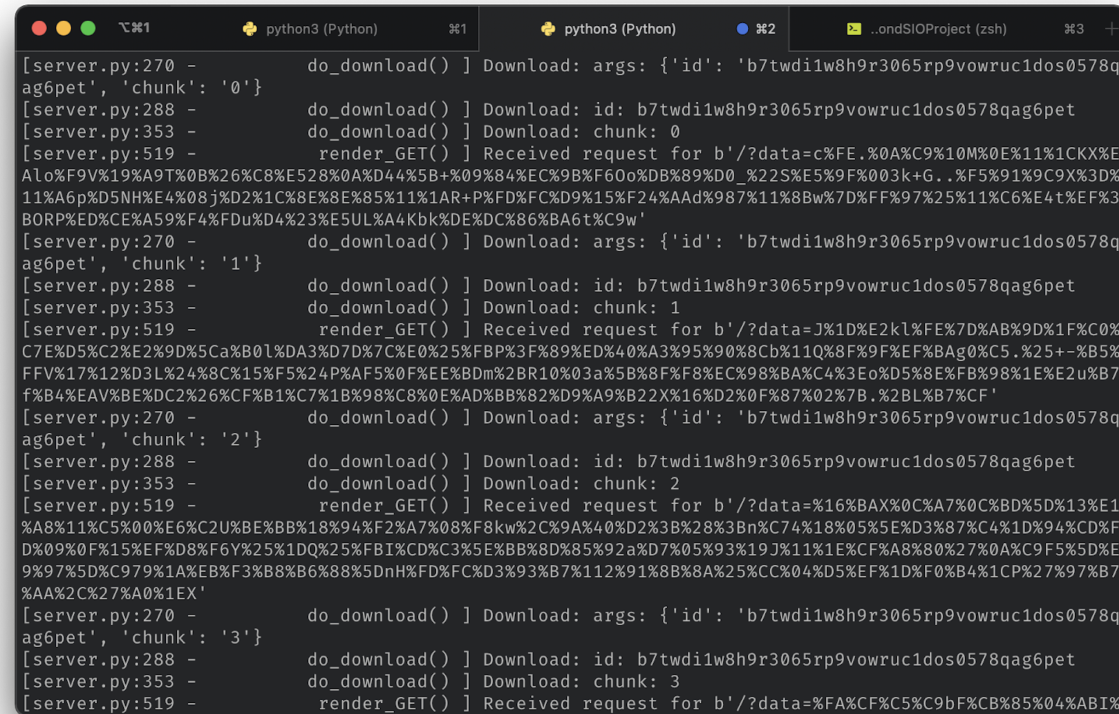


fig 7. server chunk download

To download a music the client will send a request with the chunks id in ascending order.

References

https://www.cloudflare.com/learning/ssl/what-happens-in-a-tls-handshake/

https://www.acunetix.com/blog/articles/tls-ssl-cipher-hardening/

https://thecybersecurityman.com/2018/04/25/https-the-tls-handshake-using-diffie-hellman-ephemeral/

https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.0.0/com.ibm.mq.sec.doc/q009930_.htm

https://www.thesslstore.com/blog/explaining-ssl-handshake/

https://www.cloudflare.com/learning/ssl/what-is-a-session-key/

https://docs.twistedmatrix.com/en/twisted-18.9.0/web/howto/using-twistedweb.html