universidade de aveiro
theoria poiesis praxis

SEGURANÇA INFORMÁTICA NAS ORGANIZAÇÕES

DIGITAL RIGHTS MANAGEMENT

Desenvolvido por:

Eleandro Laureano: 83069
Nuno Matamba: 78444

INTRODUÇÃO

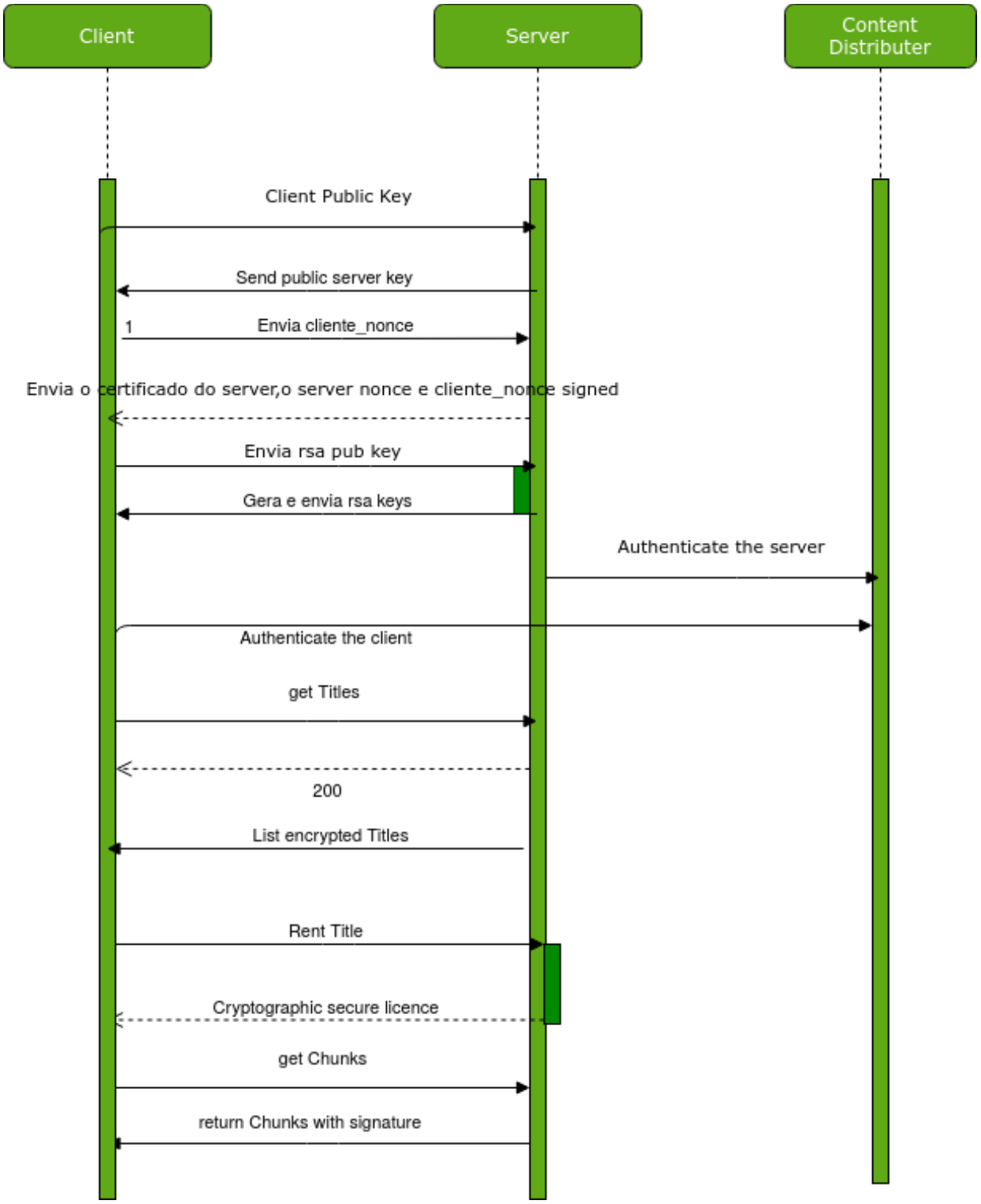
O nosso projeto, proposto no âmbito da disciplina de *Segurança Informática Nas Organizações* consiste na implementação de um protocolo de segurança e autenticação entre um canal de comunicação seguro entre um cliente e um servidor, para a possibilidade de aluguer de *media titles*.

O projeto, *Digital Rights Management* tem como objetivo a negociação entre as duas entidades para a decisão do algoritmo de encriptação, modo de operação e a síntese a utilizar para a encriptação dos "chunks" dos títulos a enviar sucessivamente para os clientes tendo varias opções para cada um dos elementos.

Utilizámos o algoritmo de Diffie-Hellman para transferências de maneira segura a(s) key(s) necessária(s) para a encriptação e desencriptação das mensagens.

Este relatório tem como objetivo a explicação do estudo de conceitos relacionados com o estabelecimento de uma sessão segura entre duas entidades e conceitos relacionados com a segurança na troca de chaves, cifras simétricas e controlo de integridade.

DIAGRAMA


Client
Server
Content Distributer
Client Public Key
Send public server key
1
Envia cliente_nonce
Envia o certificado do server,o server nonce e cliente_nonce signed
Envia rsa pub key
Gera e envia rsa keys
Authenticate the server
Authenticate the client
get Titles
200
List encrypted Titles
Rent Title
Cryptographic secure licence
get Chunks
return Chunks with signature

CONFIDENCIALIDADE E INTEGRIDADE

A sessão entre o cliente e o servidor começa com a negociação de algoritmos utilizando uma comunicação segura, implicando o uso de:

- Algoritmo de cifra
- Modo de cifra
- Função de síntese

Onde temos pelo menos, 3 algoritmos de cifra, 3 modos de cifra e 3 funções de síntese.

```
algs = ['AES', '3DES', 'ChaCha20']
mods = ['ECB', 'CFB', 'OFB']
digest_algorithms = ['SHA256', 'SHA512', 'SHA3256']
```

Em seguida, esta próxima negociação de chaves é feita através do algoritmo **Diffie Hellman.** Começamos por ter 2 valores gerados, tanto do parâmetro *g*, como do *p*.

```
def dh_generate_parameters(key_size=2048):
    parameters = dh.generate_parameters(generator=2, key_size=key_size)
    parameter_numbers = parameters.parameter_numbers()
    p = parameter_numbers.p
    g = parameter_numbers.g

    return [p,g]
```

Posteriormente, para se estabelecer a ligação **server-client,** faz-se uma troca de chaves **Diffie-Hellman** onde cada um deles faz um *handshake*, mantendo assim um segredo em comum

A seguir, o cliente envia para o servidor a sua *public key* e posteriormente recebe a *public key* do servidor.

```
dh_private_k = diffie_hellman.diffie_hellman_generate_private_key(dh_parameters)
dh_public_num = diffie_hellman.diffie_hellman_generate_public_key(dh_private_k)

#getting the server public number
req = requests.post(f'{SERVER_URL}/api/dh-handshake', data=json.dumps([dh_public_num]).encode('latin'))
if req.status_code == 200:
    print("Got server public number")
```

Posteriormente, o servidor gera uma *private key* para ser usada durante o intercâmbio.
E após o processo entre o cliente e o servidor ser completado, conclui-se assim o processo de negociação de chaves entre os mesmos.

```
    """Generate a private key, a public key and public number of client"""
    public_number_of_client = json.loads(request.content.read())[0]
    self.diffie_hellman_private_key = diffie_hellman_generate_private_key(self.dh_parameters)
    diffie_hellman_public_number = diffie_hellman_generate_public_key(self.diffie_hellman_private_key)
    self.secret_key = diffie_hellman_common_secret(self.diffie_hellman_private_key, public_number_of_client)
    request.responseHeaders.addRawHeader(b"content-type", b"application/json")
    return json.dumps([diffie_hellman_public_number], indent=4).encode('latin')
```

Assim sendo, para garantir a confidencialidade das comunicações entre o cliente e o servidor ao encriptar-se as mesmas, temos uma função chamada *encrypt* para esse mesmo propósito.

```
"""This function is used to encrypt the data"""
def encrypt(password, message, algorithm_name, cipherMode=None):
    if type(message) != type(b""):
        message = message.encode()

    salt = os.urandom(16)

    key = generate_key(algorithm_name, salt, password)

    if algorithm_name == 'ChaCha20':
        nonce = token_bytes(16)
        algorithm = algorithms.ChaCha20(key, nonce)
        block_length = 128
    elif algorithm_name == '3DES':
        block_length = 8
        algorithm = algorithms.TripleDES(key)
    else:
        block_length = 16
        algorithm = algorithms.AES(key)

    iv = None
    if algorithm_name != "ChaCha20" and cipherMode != "ECB":
        iv = token_bytes(block_length)

    if cipherMode == "CFB":
        cipher_mode = modes.CFB(iv)
    elif cipherMode == "OFB":
        cipher_mode = modes.OFB(iv)
    elif cipherMode == "ECB":
        cipher_mode = modes.ECB()
    else:
```

Para a prevenção de utilizadores abusarem do sistema, o **server** necessita de ter conhecimento se os clientes ainda o podem utilizar e para isso, utilizamos licenças com um tempo limite.

E agora, quanto a gestão das licenças com base no tempo e o número de views, temos um dicionário chamado *licenses* para nos ajudar a estipular o tempo concedendo a licenças 5 minutos como tempo máximo para ter um *media file* disponível. Cada utilizador somente tem acesso a uma licença válida. Na perspetiva do **server**, antes de enviar *chunks* ao **client,** verifica se o **client** possui uma licença válida.

Sendo assim, somente com as licenças válidas, o **client** consegue a reprodução dos ficheiros.

```
for item in media_list:
    d = datetime.now() + timedelta(minutes=5)
    print("TEMPO", d)
    licenses[index] = [1, d.timestamp()]
    print(f'{index} - {media_list[index]["name"]}')
    index += index
    print("Index", index)
```

```
selection = int(selection)
if 0 <= selection < len(media_list):
    if licenses.get(int(selection))[0] == 0 or datetime.now().timestamp() > licenses.get(int(selection))[1]:
        continue
    licenses[selection][0] -= 1
    print(licenses.get(int(selection))[0])
    break
```

AUTENTICAÇÃO E ISOLAMENTO

No que diz respeito a autenticação e isolamento, começa-se pelo facto de que o **server** recebe um *nonce* gerado pelo **client**, em que o **server** assina com a sua chave privada e envia para o **client**.

```
client_nonce = os.urandom(64)
encripted_client_nonce = secure.encrypt(secret_key,client_nonce,
cipher_list[0],cipher_list[1]).decode('latin')
req = requests.post(f'{SERVER_URL}/api/server_auth',data = json.dumps({"nonce":encripted_client_nonce}).encode())
```

Entretanto, utilizando a aplicação **XCA** conseguimos criar um *CA* de raiz e um certificado para o **server**, que é validado por esse mesmo *CA*. Após isso, para o **client** autenticar o **server**, o próprio **server** envia o seu certificado ao **client** e então o **client** verifica a validade do seu certificado.

Em seguida, quanto ao conteúdo consumido, realiza-se uma autenticação do mesmo através da assinatura do conteúdo com chaves **RSA** e após isso é realizada uma troca de chaves.

Caso a mensagem não esteja assinada pelo **server**, isso indica que o **client** não está autenticado, recebendo assim uma mensagem de erro.

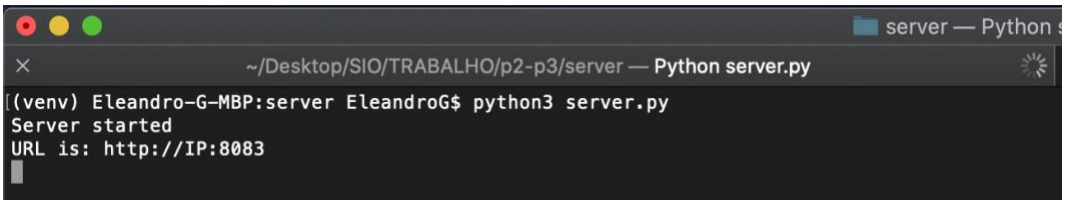
Então, o ficheiro é enviado para o **client** **“**chunk by chunk” e para a prevenção de ataques **Man In The Middle**, decidimos autenticar cada *chunk* enviado e também o **server** realiza a sua encriptação.

Depois, sempre que é feito um pedido de *get()* de um *chunk*, o **server** gera uma assinatura com a sua chave privada para esse *chunk* e irá juntar à mensagem a ser encaminhada para o **client**. O **client** ao receber o *chunk*, verifica se assinatura do **server** é válida e caso contrário, envia uma mensagem a dizer que não é confiável.

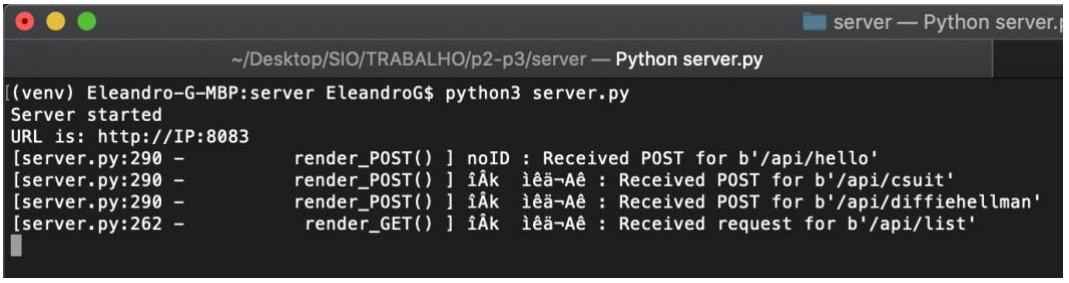
Sendo assim, ele somente realizada o *decrypt()* e faz *load()* do seu conteúdo caso seja válida.

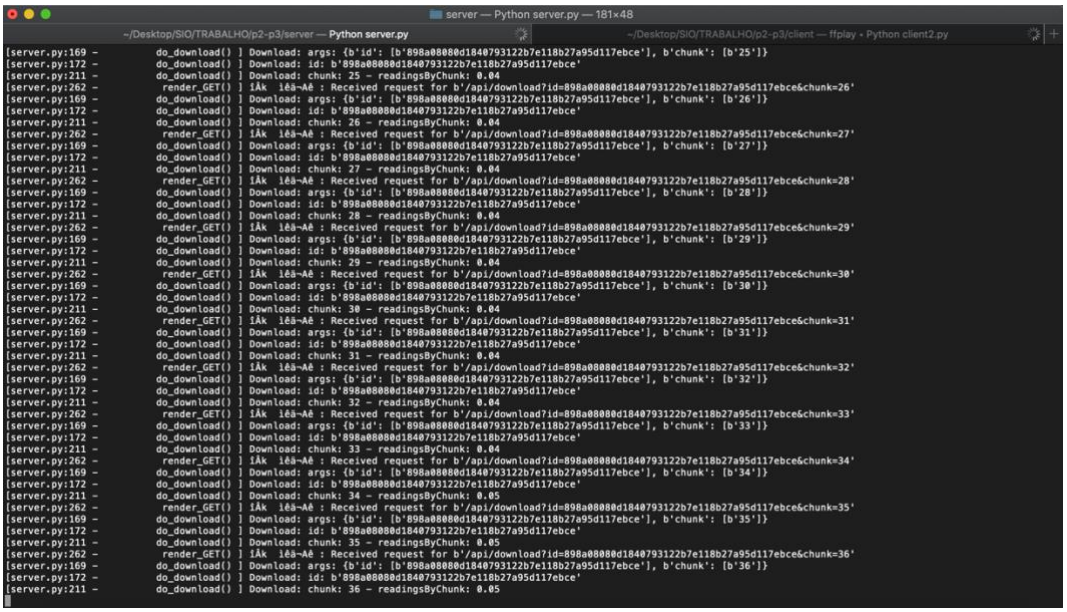
```
for chunk in range(media_item['chunks'] + 1):
    """Decrypt based on key rotation"""
    req = requests.get(f'{SERVER_URL}/api/download?id={media_item["id"]}&chunk={chunk}')

    if not req.status_code == 200:
        print(secure.decrypt(secret_key,req.json()['error'],cipher_list[0],cipher_list[1]).decode())

    chunk = req.json()

    data = binascii.a2b_base64(secure.decrypt(secret_key,chunk['data'].encode('latin'),
    cipher_list[0],cipher_list[1]))
    data_signature = secure.decrypt(secret_key,chunk['data_signature'], cipher_list[0],cipher_list[1])

    if not rsa.rsa_verify(rsa.load_rsa_public_key("../rsa_keys/server_rsa_pub_key.pub"), data,data_signature):
        print("The file sent from the server is not of trust")
        sys.exit(0)
    print("Chunk has a valid signature")
```

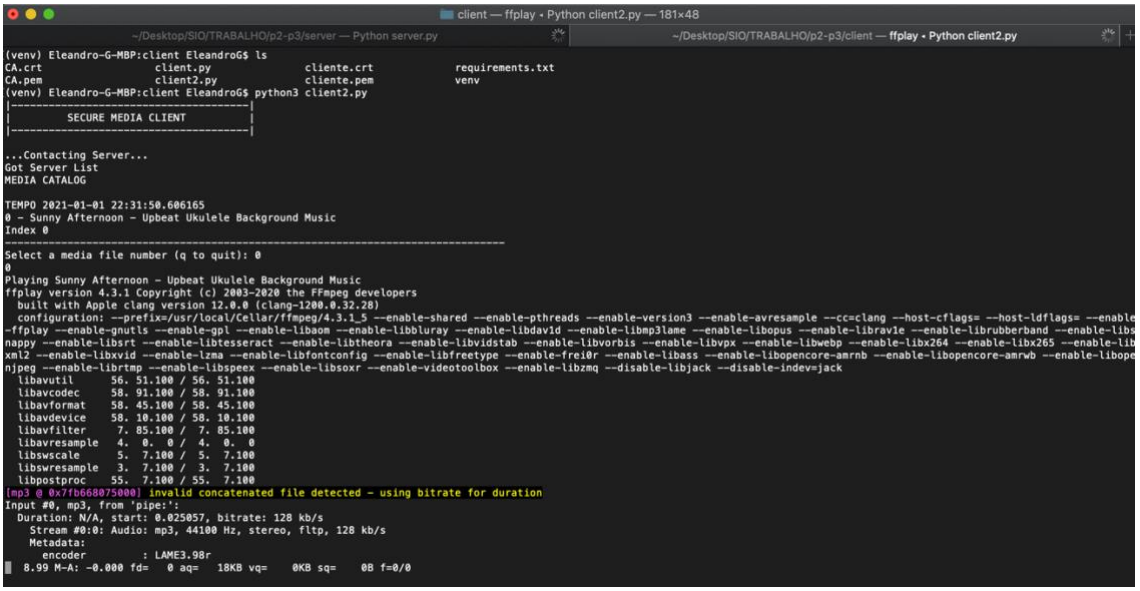
Execução

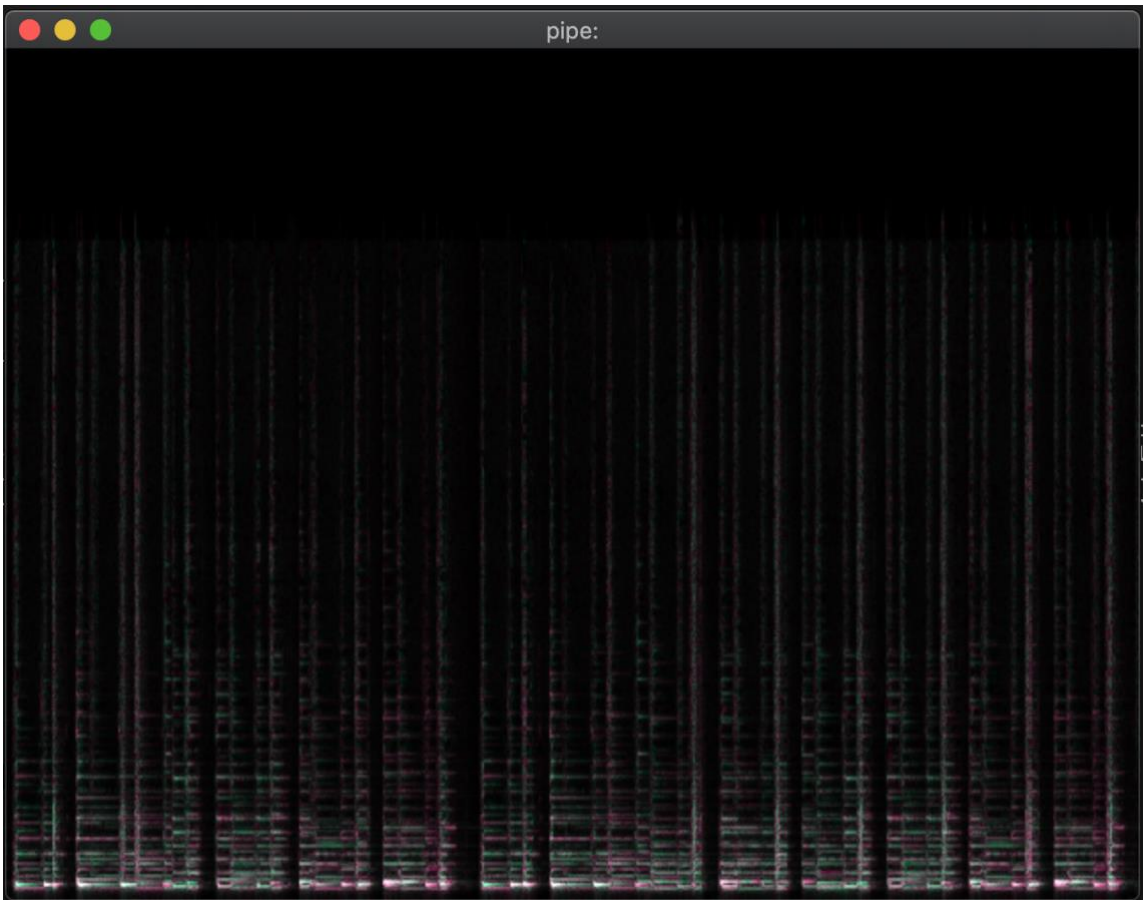



```
Eleandro-G-MBP:client EleandroG$ python3 client.py
Got dh-parameters
Got server public number
Got the key to encrypt communication
Got ciphers list
Chosen ciphers:
        Algorithm: AES
        Cipher Mode: OFB
        Hash Function: SHA512
Got server encrypted message
server message:  The Cryptography Pattern was established
Received Certificated and Signed Nonce
Certificated Chain is not completed
Received Server public rsa key
|-------------------------------------|
|         SECURE MEDIA CLIENT         |
|-------------------------------------|

...Contacting Server...
Got Server List
MEDIA CATALOG

TEMPO 2021-02-25 18:49:06.079836
0 - Sunny Afternoon - Upbeat Ukulele Background Music
Index 0
--------------------------------------------------------------------------------
Select a media file number (q to quit):
```




server — Python server.py — 181×48
~/Desktop/SIO/TRABALHO/p2-p3/server — Python server.py
~/Desktop/SIO/TRABALHO/p2-p3/client — ffplay • Python client2.py
[server.py:169 - do_download()] Download: args: {b'id': [b'898a08080d1840793122b7e118b27a95d117ebce'], b'chunk': [b'25']}
[server.py:172 - do_download()] Download: id: b'898a08080d1840793122b7e118b27a95d117ebce'
[server.py:211 - do_download()] Download: chunk: 25 - readingsByChunk: 0.04
[server.py:262 - render_GET()] ÍÂÃ ìêà~Aê : Received request for b'/api/download?id=898a08080d1840793122b7e118b27a95d117ebce&chunk=26'
[server.py:169 - do_download()] Download: args: {b'id': [b'898a08080d1840793122b7e118b27a95d117ebce'], b'chunk': [b'26']}
[server.py:172 - do_download()] Download: id: b'898a08080d1840793122b7e118b27a95d117ebce'
[server.py:211 - do_download()] Download: chunk: 26 - readingsByChunk: 0.04
[server.py:262 - render_GET()] ÍÂÃ ìêà~Aê : Received request for b'/api/download?id=898a08080d1840793122b7e118b27a95d117ebce&chunk=27'
[server.py:169 - do_download()] Download: args: {b'id': [b'898a08080d1840793122b7e118b27a95d117ebce'], b'chunk': [b'27']}
[server.py:172 - do_download()] Download: id: b'898a08080d1840793122b7e118b27a95d117ebce'
[server.py:211 - do_download()] Download: chunk: 27 - readingsByChunk: 0.04
[server.py:262 - render_GET()] ÍÂÃ ìêà~Aê : Received request for b'/api/download?id=898a08080d1840793122b7e118b27a95d117ebce&chunk=28'
[server.py:169 - do_download()] Download: args: {b'id': [b'898a08080d1840793122b7e118b27a95d117ebce'], b'chunk': [b'28']}
[server.py:172 - do_download()] Download: id: b'898a08080d1840793122b7e118b27a95d117ebce'
[server.py:211 - do_download()] Download: chunk: 28 - readingsByChunk: 0.04
[server.py:262 - render_GET()] ÍÂÃ ìêà~Aê : Received request for b'/api/download?id=898a08080d1840793122b7e118b27a95d117ebce&chunk=29'
[server.py:169 - do_download()] Download: args: {b'id': [b'898a08080d1840793122b7e118b27a95d117ebce'], b'chunk': [b'29']}
[server.py:172 - do_download()] Download: id: b'898a08080d1840793122b7e118b27a95d117ebce'
[server.py:211 - do_download()] Download: chunk: 29 - readingsByChunk: 0.04
[server.py:262 - render_GET()] ÍÂÃ ìêà~Aê : Received request for b'/api/download?id=898a08080d1840793122b7e118b27a95d117ebce&chunk=30'
[server.py:169 - do_download()] Download: args: {b'id': [b'898a08080d1840793122b7e118b27a95d117ebce'], b'chunk': [b'30']}
[server.py:172 - do_download()] Download: id: b'898a08080d1840793122b7e118b27a95d117ebce'
[server.py:211 - do_download()] Download: chunk: 30 - readingsByChunk: 0.04
[server.py:262 - render_GET()] ÍÂÃ ìêà~Aê : Received request for b'/api/download?id=898a08080d1840793122b7e118b27a95d117ebce&chunk=31'
[server.py:169 - do_download()] Download: args: {b'id': [b'898a08080d1840793122b7e118b27a95d117ebce'], b'chunk': [b'31']}
[server.py:172 - do_download()] Download: id: b'898a08080d1840793122b7e118b27a95d117ebce'
[server.py:211 - do_download()] Download: chunk: 31 - readingsByChunk: 0.04
[server.py:262 - render_GET()] ÍÂÃ ìêà~Aê : Received request for b'/api/download?id=898a08080d1840793122b7e118b27a95d117ebce&chunk=32'
[server.py:169 - do_download()] Download: args: {b'id': [b'898a08080d1840793122b7e118b27a95d117ebce'], b'chunk': [b'32']}
[server.py:172 - do_download()] Download: id: b'898a08080d1840793122b7e118b27a95d117ebce'
[server.py:211 - do_download()] Download: chunk: 32 - readingsByChunk: 0.04
[server.py:262 - render_GET()] ÍÂÃ ìêà~Aê : Received request for b'/api/download?id=898a08080d1840793122b7e118b27a95d117ebce&chunk=33'
[server.py:169 - do_download()] Download: args: {b'id': [b'898a08080d1840793122b7e118b27a95d117ebce'], b'chunk': [b'33']}
[server.py:172 - do_download()] Download: id: b'898a08080d1840793122b7e118b27a95d117ebce'
[server.py:211 - do_download()] Download: chunk: 33 - readingsByChunk: 0.04
[server.py:262 - render_GET()] ÍÂÃ ìêà~Aê : Received request for b'/api/download?id=898a08080d1840793122b7e118b27a95d117ebce&chunk=34'
[server.py:169 - do_download()] Download: args: {b'id': [b'898a08080d1840793122b7e118b27a95d117ebce'], b'chunk': [b'34']}
[server.py:172 - do_download()] Download: id: b'898a08080d1840793122b7e118b27a95d117ebce'
[server.py:211 - do_download()] Download: chunk: 34 - readingsByChunk: 0.05
[server.py:262 - render_GET()] ÍÂÃ ìêà~Aê : Received request for b'/api/download?id=898a08080d1840793122b7e118b27a95d117ebce&chunk=35'
[server.py:169 - do_download()] Download: args: {b'id': [b'898a08080d1840793122b7e118b27a95d117ebce'], b'chunk': [b'35']}
[server.py:172 - do_download()] Download: id: b'898a08080d1840793122b7e118b27a95d117ebce'
[server.py:211 - do_download()] Download: chunk: 35 - readingsByChunk: 0.05
[server.py:262 - render_GET()] ÍÂÃ ìêà~Aê : Received request for b'/api/download?id=898a08080d1840793122b7e118b27a95d117ebce&chunk=36'
[server.py:169 - do_download()] Download: args: {b'id': [b'898a08080d1840793122b7e118b27a95d117ebce'], b'chunk': [b'36']}
[server.py:172 - do_download()] Download: id: b'898a08080d1840793122b7e118b27a95d117ebce'
[server.py:211 - do_download()] Download: chunk: 36 - readingsByChunk: 0.05


client — ffplay • Python client2.py — 181×48
~/Desktop/SIO/TRABALHO/p2-p3/server — Python server.py
~/Desktop/SIO/TRABALHO/p2-p3/client — ffplay • Python client2.py
(venv) Eleandro-G-MBP:client EleandroG$ ls
CA.crt client.py cliente.crt requirements.txt
CA.pem client2.py cliente.pem venv
(venv) Eleandro-G-MBP:client EleandroG$ python3 client2.py
|--------------------------------------|
SECURE MEDIA CLIENT
...Contacting Server...
Got Server List
MEDIA CATALOG
TEMPO 2021-01-01 22:31:50.606165
0 - Sunny Afternoon - Upbeat Ukulele Background Music
Index 0
Select a media file number (q to quit): 0
0
Playing Sunny Afternoon - Upbeat Ukulele Background Music
ffplay version 4.3.1 Copyright (c) 2003-2020 the FFmpeg developers
built with Apple clang version 12.0.0 (clang-1200.0.32.28)
configuration: --prefix=/usr/local/Cellar/ffmpeg/4.3.1_5 --enable-shared --enable-pthreads --enable-version3 --enable-avresample --cc=clang --host-cflags= --host-ldflags= --enable
-ffplay --enable-gnutls --enable-gpl --enable-libaom --enable-libbluray --enable-libdav1d --enable-libmp3lame --enable-libopus --enable-librav1e --enable-librubberband --enable-libs
nappy --enable-libsrt --enable-libtesseract --enable-libtheora --enable-libvidstab --enable-libvorbis --enable-libvpx --enable-libwebp --enable-libx264 --enable-libx265 --enable-lib
xml2 --enable-libxvid --enable-lzma --enable-libfontconfig --enable-libfreetype --enable-frei0r --enable-libass --enable-libopencore-amrnb --enable-libopencore-amrwb --enable-libope
njpeg --enable-librtmp --enable-libspeex --enable-libsoxr --enable-videotoolbox --enable-libzmq --disable-libjack --disable-indev=jack
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 @ 0x7fb668075000] invalid concatenated file detected - using bitrate for duration
Input #0, mp3, from 'pipe:':
Duration: N/A, start: 0.025057, bitrate: 128 kb/s
Stream #0:0: Audio: mp3, 44100 Hz, stereo, fltp, 128 kb/s
Metadata:
encoder : LAME3.98r
8.99 M-A: -0.000 fd= 0 aq= 18KB vq= 0KB sq= 0B f=0/0




```
Select a media file number (q to quit): q
(venv) Eleandro-G-MBP:client EleandroG$
```

```
[server.py:262 -           render_GET() ] îÂk  ìêä¬Aê : Received request for b'/api/list'
[server.py:290 -          render_POST() ] îÂk  ìêä¬Aê : Received POST for b'/api/bye'
```