Seguridad y gestión de derechos digitales

Proyecto Final. Sistema de protección de contenidos digitales

INTEGRANTES:

- *Álvaro Donderis Martínez*
- *Dazhan Hong*
- *Marcos Olmo López*
- *Pablo Colomino Granell*

ÍNDICE

INTRODUCCIÓN:

En el creciente entorno actual de la distribución digital de contenido, proteger la propiedad intelectual y garantizar un acceso controlado son desafíos clave para creadores, distribuidores y consumidores. Este proyecto presenta un sistema de protección de contenidos digitales desarrollado en Python, que emplea una arquitectura robusta compuesta por un servidor de contenidos, un servidor de licencias, una Aplicación de Usuario (UA) y un Módulo de Descripción de Contenido (CDM).

La seguridad del sistema se logra mediante el uso de criptografía moderna: toda comunicación entre los componentes está protegida con cifrado simétrico AES en modo CBC (Cipher Block Chaining), mientras que la clave de cifrado se intercambia utilizando criptografía asimétrica RSA. Este enfoque asegura tanto la confidencialidad como la integridad de los datos transmitidos, protegiendo el contenido digital contra accesos no autorizados y asegurando su correcta distribución a los usuarios autorizados.

El sistema busca abordar las necesidades fundamentales de la gestión de derechos digitales (DRM), tales como el control de acceso y la distribución segura de claves. A través de este trabajo, se demuestra la viabilidad de implementar una solución de DRM eficiente y segura utilizando herramientas sencillas; en este caso, el entorno de desarrollo Thonny junto con la librería 'socket' de Python, además de otras para poder tratar correctamente con todos los contenidos.

Figura 1. Concepto de sistema de protección de contenidos digitales

Nuestra implementación del sistema sigue el esquema de comunicación de la Figura 2 entre los distintos programas, estando gran parte de la comunicación encriptada.

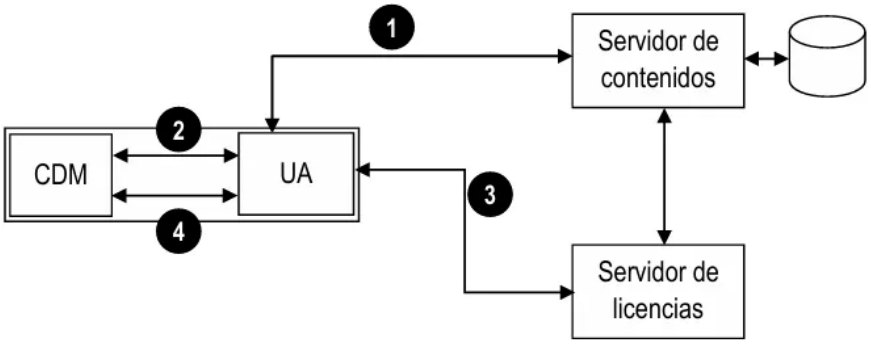

Figura 2. Diagrama de flujo entre programas

Estando el flujo del programa dividido en 4 fases:

- Fase I: Petición y envío del contenido desde el servidor de contenidos a la UA, junto con una marca de agua visible que identifica al usuario que solicitó el contenido si este es una imagen.
- Fase II: Petición de una solicitud de licencia de UA y envío de CDM de la solicitud de licencia para el servidor de licencias junto con una firma digital para autenticar al usuario.
- Fase III: Petición y envío de la licencia necesaria para desencriptar el contenido inicial alojado en el servidor.
- Fase IV: Desencriptación del contenido mediante licencia en CDM y envío a UA, que almacena los contenidos descifrados en el equipo.

DESARROLLO:

Servidor de contenidos:

El servidor de contenidos de esta aplicación guarda vídeos e imágenes en un sistema de directorios que los divide, a parte de por tipo de archivo, en cifrados y no cifrados, para saber si se han de transmitir de forma segura a la aplicación de usuario. A su vez, un registro de claves y vectores iniciales con los que cifrar los contenidos necesarios está guardado en un fichero JSON capaz de ser actualizado por el servidor.


```
{
    "enviar/imagen_cifrada1.txt": {
        "clave": "04e596f391ebb0c76afb557fb08c0e2b",
        "iv": "becf53d52115a8ee2aaa0dd20da38958"
    },
    "enviar/imagen_cifrada2.txt": {
        "clave": "f4b605f773a78674623c316bcddfd942",
        "iv": "7213bd5a556c8977b74be809d8b3e5ad"
    },
    "enviar/video_cifrado1.txt": {
        "clave": "f60ca8ff1fa8b8790809d8d6b6fdd9ba",
        "iv": "2afb6d8061d005499c3b5e7f1c701849"
    },
    "enviar/video_cifrado2.txt": {
        "clave": "9f2bcb4da0aff927674f793439c48ec4",
        "iv": "3d4a3ed1b29db9c61dee9e9470644548"
    }
}
```


Figura 3. Base de datos JSON

El código del programa incluye funciones básicas para encriptar y desencriptar textos y archivos de texto con AES CBC accediendo al registro de claves, para poder transmitir contenidos cifrados. La transmisión de estos se hará guardando la información en texto plano y enviándolo. Por seguridad, antes de hacerlo, se almacenará en archivos de texto que se eliminarán más tarde. Todos los archivos, de hecho, se convierten a texto para ser compartidos. Existen, con este objetivo en mente, funciones para transformar en texto imágenes y vídeos, usando el paquete “base 64”.

Para poder cubrir la necesidad de añadir marcas de agua visibles a las imágenes que se vayan a enviar, existe la función *marcaAgua_visible,* que recibe la ruta de un archivo como *input*, otra como *output* donde dejará la nueva imagen y un *ID* que añadirá como marca de agua.

La función principal del programa es la de *servidor* que maneja la comunicación con clientes usando el método *select* y contiene, a su vez, otras funciones auxiliares y variables propias. Estas sirven para: enviar mensajes y archivos, borrar el contenido de un directorio, listar el contenido de una carpeta y enviar un aviso al cliente de qué tipo de archivo y si va a estar encriptado se le enviará.

Por otra parte, el manejo de las peticiones de los clientes se hace distinguiendo entre si piden listar los contenidos disponibles, si quieren cerrar conexión, o si quieren que se les envíe un archivo. No existe mucho misterio en los dos primeros casos, sin embargo, con la última opción se ha de comprobar si el contenido está en el sistema de directorios, y si lo hace saber si debe cifrarse o añadirse marca de agua. Antes de ser enviado como texto, aún así, hay más. Al cliente hay que decirle, como ya se ha mencionado, el tipo de información y si está encriptada, pero además, también hay que advertir del tamaño total del mensaje. Cuando todo el proceso termine los ficheros que estén en el directorio *enviar* serán eliminados. Este directorio se usa de forma temporal cada vez que hay un pedido para poder dejar los archivos que se van transformando: las imágenes con marca de agua y los textos que contienen la información a enviar.

```
list
Contenido Disponible:
imagen1.png
imagen2.png
video1.mp4
video2.mp4
imagen_cifrada1.png
imagen_cifrada2.png
video_cifrado1.mp4
video_cifrado2.mp4

Mensaje enviado
```

Figura 4. Lista de contenido llamada con 'list'

```
Introduce el nombre del archivo que deseas descargar:imagen1.png
```

Figura 5. Consola de UA donde se introduce el contenido deseado

Figura 6. Imagen con marca de agua visible (esquina superior izquierda)

Servidor de licencias:

1. Generación de claves y cifrado

generar_claves(): Crea una clave AES y un IV (vector de inicialización) aleatorios para cifrar datos usando el modo AES-CBC.

cifrar_contenido() y *descifrar_contenido()*: Usan el algoritmo AES en modo CBC para cifrar y descifrar datos. Incluyen padding (relleno) para asegurar que el tamaño de los datos sea múltiplo del tamaño del bloque (128 bits).

cifrar_rsa_bytes(): Implementa cifrado RSA para mensajes usando las claves públicas proporcionadas. Convierte cada byte del mensaje en un entero y lo cifra con la fórmula $c = m^e \ mod \ n$, donde *e* y *n* son parte de la clave pública RSA.

2. Comunicación TCP

des_comunicacion(): Establece un servidor TCP para esperar conexiones de un cliente (app o dispositivo).

Recibe la clave pública RSA del cliente (valores e y n) para realizar el cifrado RSA de las claves de comunicación.

Genera nuevas claves AES para la comunicación (*clave_com* e *iv_com*) y las cifra usando la clave pública del cliente.

Envía las claves cifradas al cliente para que ambos puedan usar AES para el intercambio posterior de información.

procesar_datos(): Descifra el mensaje con AES y valida los datos recibidos del cliente. Separa el contenido y la firma digital y verifica la firma digital usando RSA para asegurar que el mensaje no ha sido alterado.

Busca el contenido solicitado en el archivo JSON cifrado, lo descifra y reenvía las claves e IV del contenido solicitado al cliente, cifrándolas nuevamente con las claves de comunicación.

3. Seguridad con JSON cifrado

cifrar_json(): Cifra un archivo JSON que contiene claves e IV asociados a diferentes contenidos usando AES para cifrar cada clave e IV del JSON y guarda los datos cifrados en un nuevo archivo.

El archivo JSON original (claves_contenido.json) contiene claves e IV en texto plano, pero el archivo cifrado (claves_contenido_cifradas.json) almacena esta información en formato hexadecimal después de cifrarla con AES.

4. Firma digital

comprobar_firma(): Usa RSA para verificar la autenticidad de una firma digital. Esto asegura que los datos provienen del remitente esperado y no han sido alterados.

Flujo del codigo:

Aqui se implementa una combinación de **cifrado, autenticación y comunicación TCP** para el intercambio seguro de claves e información entre el servidor de licencias y la aplicación.

Inicialmente se generan una clave maestra y un IV maestro. Se cifra el archivo *claves_contenido.json* con esta clave e IV y se guarda como *claves_contenido_cifradas.json*.

El servidor escucha en el puerto 8889, esperando una conexión del cliente. A continuación, el cliente envía su clave pública RSA y el servidor genera claves AES para la comunicación, las cifra con RSA y las envía al cliente.

El cliente envía un mensaje cifrado solicitando un contenido específico del JSON cifrado que posee inicialmente el servidor de licencias, además de una licencia para validar su identidad. El servidor valida la firma, encuentra la clave e IV asociadas al contenido y las cifra con las claves de comunicación antes de enviarlas al cliente.

Las claves e IV solicitadas se envían al cliente en formato cifrado, garantizando seguridad durante la transferencia.

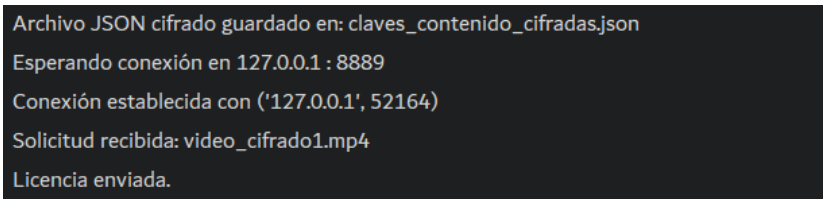

Figura 7. Consola del servidor de licencias

UA (User Application):

1. Comunicación con servidor de contenidos

recibir_contenido(): Establece la conexión con el servidor de contenidos y el usuario puede introducir el comando “list” para obtener la lista de contenidos o introducir el archivo que desea descargar.

En caso de que el usuario quiere descargar un archivo, envía la solicitud al servidor de contenidos y recibe el “tipo” del contenido. Esta variable identifica si el contenido está cifrado o no, e incluye el tipo de archivo (imagen o video). Como esta función tiene un bucle infinito, si el usuario introduce algo erróneo, se indicará en el terminal y puede volver a introducir texto.

Una vez recibida la variable “tipo”, utilizamos la función *verificar_cifrado()* para verificar si el contenido que desea descargar está cifrado o no.

Después de la comprobación, si el contenido está en claro, la UA lo puede recibir directamente. Sin embargo, si el contenido está cifrado, tenemos que recibir en primer lugar “size_data” que es el tamaño de string del “contenido”. Según la longitud total, hacemos un bucle que comprueba cada vez que recibe un fragmento de contenido, hasta que consigue recibir el contenido completo. Por último, devuelve los variables para el uso posterior del programa.

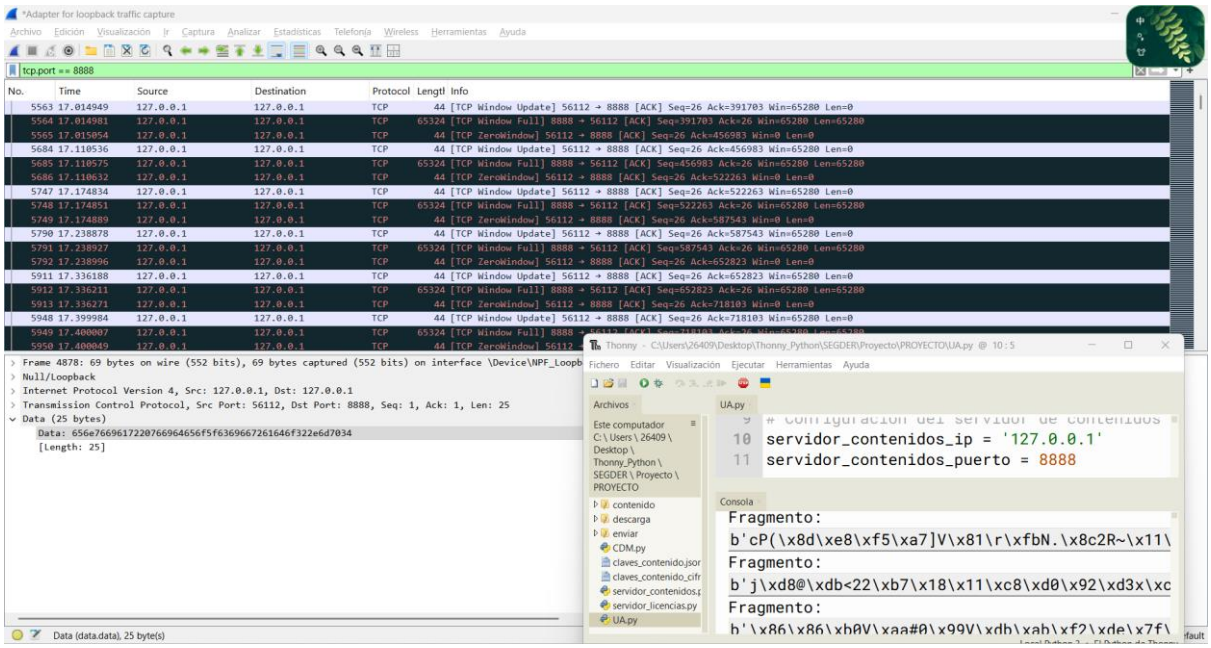

Figura 7. Intento de captura en Wireshark del contenido

2. Comunicación con CDM

En caso de que el contenido esté cifrado, la UA requiere una licencia que contenga la clave e IV que se necesitan para descifrar este -> pedir_licencia_CDM(licencia_deseada). Para conseguirla, es necesaria una solicitud de licencia firmada digitalmente, tarea de la cual se encarga el CDM. Por ello, la UA se comunica ahora con este módulo, estableciendo las claves AES de la comunicación cifrada mediante RSA primero, para recibir la solicitud de licencia firmada, junto con la clave pública necesaria para comprobar la firma.

Para realizar la comunicación segura, se usa la función recibir_clave_RSA(ip_destino, puerto_destino). Esta función se usa en la UA para crear una conexión segura con cualquier otro módulo. Necesita la IP y puerto del destino. Primero, crea una clave pública de la comunicación y una privada, se conecta a la dirección proporcionada y envía la clave pública, para que el otro módulo cifre con esta la clave de comunicación cifrada (que debe crear el otro módulo). Al haberse cifrado con la clave pública, esta clave se puede descifrar con la clave privada creada anteriormente mediante la función descifrar_rsa_bytes(clave_comunicacion_cifrada, d, n), donde 'n' es parte de la clave pública y 'd' es la privada. Esta función realiza en cada carácter de la clave cifrada la siguiente operación: pow(byte,d,n). Y así se obtiene la clave de comunicación descifrada con la que se cifrará cada mensaje que se transmita con AES en modo CBC.

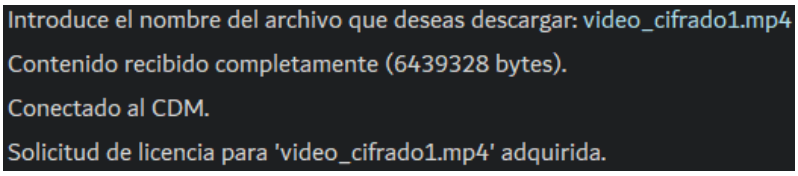

Figura 8. Consola de la UA

3. Comunicación con servidor de licencias

A continuación, se reenvía esta solicitud de licencia firmada junto con la clave pública con la que fue firmada (la del CDM) al servidor de licencias -> pedir_licencia_servidor(solicitud_licencia, clave_publica_cdm). Para ello, también se envía una clave AES para cifrar todo mensaje (mediante el cifrado asimétrico RSA explicado anteriormente).

Una vez el servidor de licencias recibe la petición y comprueba que la firma es correcta, envía la licencia (clave, IV) correspondiente a la solicitud.

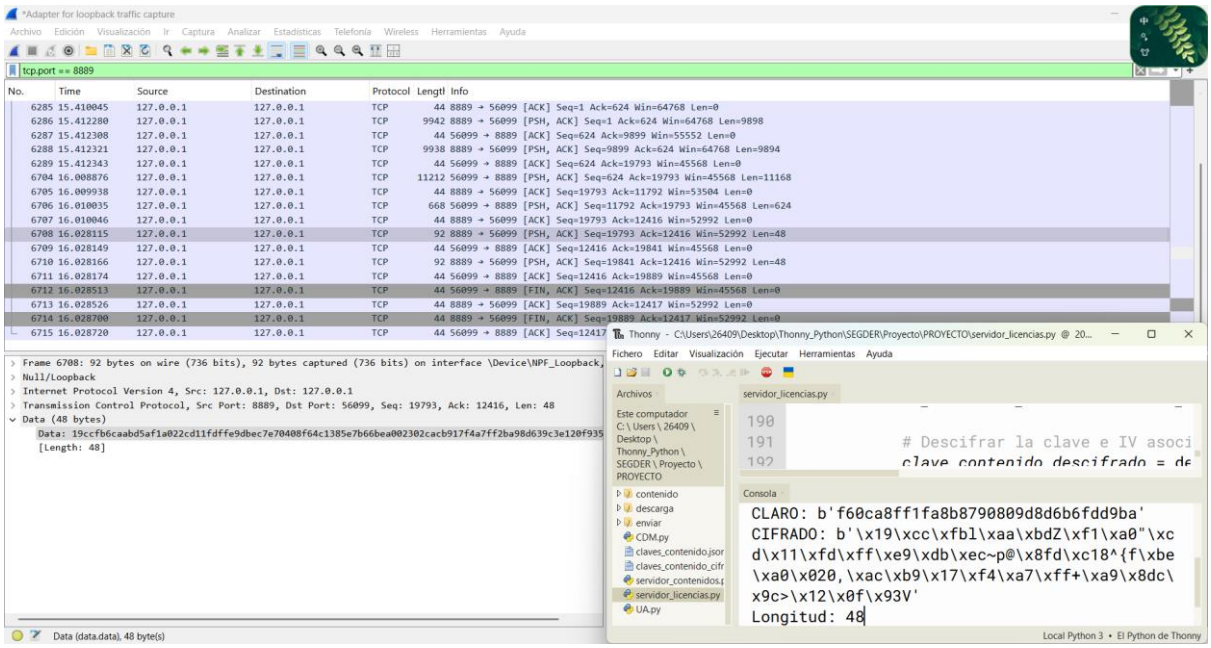

Figura 9. Intento de captura en wireshark de la clave

4. Descifrado del contenido desde CDM

descifrar_contenido_CDM(): La función tiene 2 entradas. Uno de ellos es la licencia, que contiene la clave e IV de AES en modo CBC. La otra es el contenido cifrado, que está encriptado con AES y falta desencriptar con la licencia.

En primer lugar, utilizamos la función recibir_clave_RSA() para obtener la clave e IV de la comunicación. Así podemos enviar la licencia, la longitud del contenido y el contenido en sí al CDM. Cuando CDM descifra correctamente el contenido, recibimos el contenido descifrado mediante un bucle fragmento a fragmento, igual que hemos hecho anteriormente.

Cuando ya tenemos el contenido cifrado, averiguamos el tipo de contenido y utilizamos la función correspondiente (si es imagen, utilizamos *string_to_image()* y en el caso contrario, utilizamos *string_to_video()*). La función de conversión permite convertir el String de contenido en un contenido digital y lo guarda en el directorio especificado.

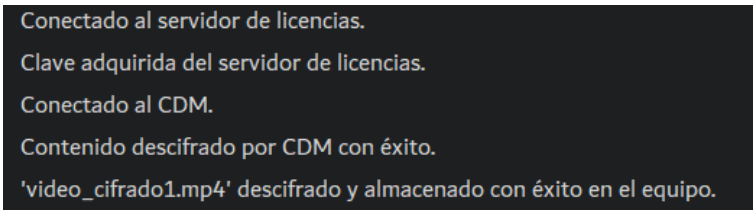

Figura 10. Consola de la UA

CDM (Content Decryption Module):

1. Comunicación UA-CDM (solicitud de licencia)

En primer lugar, se queda a la espera de la conexión con la UA. Esta se efectúa de manera segura con una función que se complementa con recibir_clave_RSA explicada anteriormente. Esta es enviar_clave_RSA(), que es llamada por recibir_peticion_ua(). Esta función recibe la clave pública de la UA y cifra la clave e IV creados con generar_claves(), que las genera de manera aleatoria. Para cifrar se usa cifrar_rsa_bytes(mensaje, e, n). Esta función realiza en cada carácter de la clave cifrada la siguiente operación: pow(byte,e,n), donde 'e' y 'n' son la clave pública. Por último, envía la clave e IV cifrados.

Mediante la función recibir_peticion_ua(), que llama a todas las explicadas anteriormente, se recibe la petición de la UA, que es el nombre del contenido deseado.

A continuación, se llama a la función enviar_solicitud_licencia_firmada(peticion_ua), que crea la solicitud de licencia. Esta está constituida por el nombre del contenido deseado junto con su firma y por la clave pública con que se firma digitalmente este nombre. Para la firma, se usa la función firmar_msg(x,d,n). Esta función realiza en cada carácter del mensaje a firmar la siguiente operación: pow(byte,d,n), donde 'd' es la clave privada de CDM y 'n' es parte de la clave pública.

Por último, se envía tanto el nombre del contenido con la firma, como la clave pública para que el servidor de licencias pueda comprobarla a la UA, que reenviará todo al servidor de licencias.

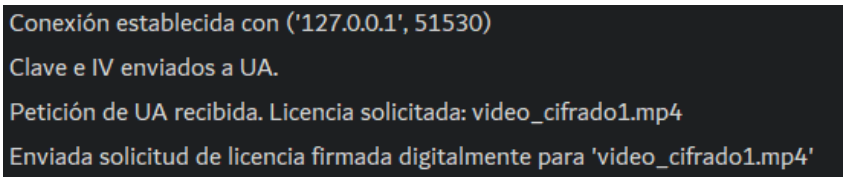

Figura 11. Consola del CDM

2. Descifrado del contenido desde CDM

Una vez obtenida la licencia, la UA la redirigirá al CDM, junto con el contenido cifrado que obtuvo del servidor de contenidos. Cabe destacar que para recibir este contenido es necesaria su longitud y el uso de un bucle, como se ha explicado previamente.

Tras realizar de nuevo las transferencias de claves necesarias para descifrar con AES todo lo recibido, incluyendo la licencia que desencripta el contenido, se procede a descifrar el contenido con la licencia proporcionada. Una vez hecho, se envía el contenido descifrado a la UA (no sin antes encriptarlo con AES), donde se pasará el texto a imagen o vídeo y se almacenará en el equipo.

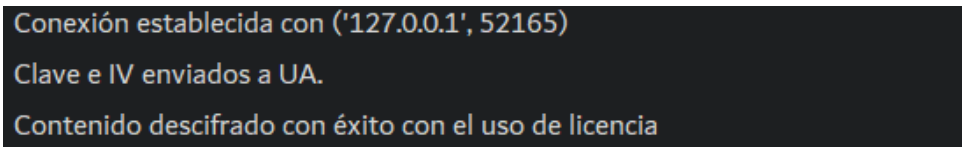

Figura 12. Consola del CDM

DESGLOSE DE TAREAS:

ÁLVARO D. M.	PABLO C. G.	DAZHAN H.	MARCOS O. L.
Servidor de Contenidos y marca de agua	Servidor de licencias	Aplicación de usuario básica (Parte I)	Adaptación de aplicación básica en UA y CDM (Parte III)
Base de datos de contenido (JSON)	Base de datos de licencias (JSON)	Recepción de contenido en UA	Firma digital
Testing y adaptaciones de compatibilidad	Vídeo explicativo del programa	Testing y adaptaciones de compatiblidad	Testing y adaptaciones de compatibilidad
Memoria	Memoria	Memoria	Memoria

CONCLUSIONES:

El sistema desarrollado cumple con los objetivos planteados al proporcionar un marco seguro y funcional para la protección de contenidos digitales. La integración de algoritmos de cifrado avanzados como AES y RSA garantiza un alto nivel de seguridad para la distribución y consumo de contenido. Además, la separación funcional entre los diferentes componentes (servidor de contenidos, servidor de licencias, UA y CDM) facilita la modularidad y escalabilidad del sistema.

El uso de bibliotecas de encriptación ha sido clave en el correcto desarrollo de la aplicación y en el cumplimiento de los objetivos de seguridad. Así como el uso del lenguaje Python, para programar de manera eficaz, aunque requiera más recursos que otros lenguajes. Este proyecto no deja de ser una simulación a un nivel pequeño de lo que son los sistemas de protección de contenido reales. No obstante, es una práctica útil e interesante para entender cómo funcionan a nivel básico y los problemas que pueden plantearse.

En conclusión, este trabajo demuestra que es posible diseñar e implementar un sistema DRM seguro y eficiente mediante el uso de técnicas criptográficas modernas. La solución propuesta puede servir como base para futuros desarrollos en la protección de contenido digital, ofreciendo un modelo que equilibra seguridad, funcionalidad y flexibilidad.

ANEXO I. CÓDIGO FUENTE

Servidor de contenidos (code):

```
20 ##############################################################
21 #                  Paquetes del programa
22 ##############################################################
23
24 from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
25 from cryptography.hazmat.primitives import padding
26 import os
27 import json
28 from PIL import Image, ImageDraw, ImageFont
29 import base64
30 from socket import *
31 import select
```

```
33 ##############################################################
34 # Funciones del programa
35 ##############################################################
36
37 # Funcion para escribir un archivo de texto
38
39 def escribir_archivo_texto(ruta,contenido):
40     with open(ruta,"w") as archivo:
41         archivo.write(contenido)
42     return None
43
```

```
44 # Función de encriptado TEXTO con AES CBC
45
46 def encriptar_texto(texto, clave, iv):
47     """
48     Cifra un texto utilizando AES en modo CBC con relleno PKCS7.
49
50     Args:
51         texto (bytes): El texto a cifrar.
52         clave (bytes): La clave de cifrado (16 bytes).
53         iv    (bytes): El vector de inicialización (16 bytes).
54
55     Returns:
56         bytes: El texto cifrado.
57     """
58     cipher = Cipher(algorithms.AES(clave), modes.CBC(iv))
59     encryptor = cipher.encryptor()
60     padder = padding.PKCS7(algorithms.AES.block_size).padder()
61     padded_data = padder.update(texto) + padder.finalize()
62     return encryptor.update(padded_data)
```

```python
# Función para generar clave e iv para un archivo y guardarlo en un JSON

def genera_clave_iv(filename,json_file="claves_contenido.json"):
    """
    Genera una clave y un iv para cifrar con AES CBC. Lo guarda en un JSON

    Args:
        filename  (string): el nombre del archivo que se cifrará con esos valores
        json_file (string): el nombre del archivo JSON donde guardar la información

    Returns:
        key, iv -> La clave y el iv generados
    """
    key = os.urandom(16).hex()
    iv = os.urandom(16).hex()

    try:
        with open(json_file, "r") as file:
            data = json.load(file)
    except FileNotFoundError:
        data = {}

    data[filename] = {"clave": key, "iv": iv}

    with open(json_file, "w") as file:
        json.dump(data, file, indent=4)

    return key, iv
```

```python
def encriptar_archivo(input_filename, output_filename, json_file="claves_contenido.json"):
    """
    Cifra un arhivo utilizando AES en modo CBC con relleno PKCS7.

    Args:
        input_filename  (string): Nombre del archivo a cifrar
        output_filename (string): Nombre de destino del archivo
        json_file       (string): Nombre del archivo JSON que contiene la clave y el iv de ese archivo

    Returns:
        contenido: El texto cifrado.
    """
    with open(json_file, "r") as file:
        data = json.load(file)

    if input_filename not in data:
        raise ValueError(f"No se encontraron clave e IV para el archivo '{input_filename}' en el JSON.")

    clave = bytes.fromhex(data[input_filename]["clave"])
    iv = bytes.fromhex(data[input_filename]["iv"])

    with open(input_filename, "rb") as file:
        texto = file.read()

    contenido = encriptar_texto(texto, clave, iv)
    try:
        escribir_archivo_texto(output_filename,contenido)
    except:
        pass
    print("\nArchivo",input_filename,"encriptado")
    return contenido
```

```python
# Funciones serializacion imagen y video

def imagen2texto(input_filename, output_filename):
    """
    Convierte un archivo de imagen PNG a texto

    Args:
        input_filename  (string): El nombre del archivo a convertir.
        output_filename (string): Nombre de destino del archivo.

    Returns:
        texto_imagen (string): El texto equivalente al contenido de la imagen
    """
    try:
        with open(input_filename, "rb") as img_file:
            texto_imagen = base64.b64encode(img_file.read()).decode('utf-8')

        escribir_archivo_texto(output_filename,texto_imagen)
        print(f"Texto de la imagen guardado en '{output_filename}'")
        return texto_imagen

    except FileNotFoundError:
        print(f"Error: El archivo '{input_filename}' no fue encontrado.")
        return None
```

```python
def video2texto(input_filename, output_filename):
    """
    Convierte un archivo de video MP4 a texto para poder ser cifrado

    Args:
        filename (string): el nombre del archivo a convertir
        output_filename (string): Nombre de destino del archivo.

    Returns:
        texto_video (string): El texto equivalente al contenido del video
    """
    try:
        with open(input_filename, "rb") as video_file:
            texto_video = base64.b64encode(video_file.read()).decode("utf-8")

        escribir_archivo_texto(output_filename,texto_video)
        print(f"Texto del vídeo guardado en '{output_filename}'")
        return texto_video

    except FileNotFoundError:
        print(f"Error: El archivo '{input_filename}' no fue encontrado.")
        return None
```

```python
def marcaAgua_visible(input_filename,output_filename, ID):
    """
    Añade una marca de agua visible a una imagen. Se creará una imagen con el contenido original añadiendo una marca de agua

    Args:
        filename        (string): El nombre de la imagen a transformar.
        output_filename (string): Nombre de destino del archivo.
        ID              (string): El identificador que se convertirá en la marca de agua visible.

    Returns:
        output_filename: Devuelve el nombre de la nueva imagen creada par apoder enviarla y destruirla
    """
    try:
        with Image.open(input_filename).convert("RGBA") as base:

            # make a blank image for the text, initialized to transparent text color
            txt = Image.new("RGBA", base.size, (255, 255, 255, 0))
            fnt = ImageFont.load_default()
            draw = ImageDraw.Draw(txt)
            draw.text((10,10),ID,font=fnt,fill=(255, 255, 255, 128))
            draw.text((10,50),ID,font=fnt,fill=(0, 0, 0, 128))

            imagen_conMarcaAgua = Image.alpha_composite(base, txt)

            imagen_conMarcaAgua.convert("RGB").save(output_filename)
            print(f"Imagen con marca de agua guardada como '{output_filename}'")

    except FileNotFoundError:
        print(f"Error: El archivo '{input_filename}' no fue encontrado.")
    except Exception as e:
        print(f"Error inesperado: {e}")
    return output_filename
```

```python
# Función para crear y gestionar el servidor

def servidor():
    try:
        # Lista de archivos de contenido disponibles
        img_public = os.listdir("contenido/publico/imagenes")
        vid_public = os.listdir("contenido/publico/videos")
        img_priv   = os.listdir("contenido/cifrado/imagenes_cifradas")
        vid_priv   = os.listdir("contenido/cifrado/videos_cifrados")
        contenido = img_public + vid_public + img_priv + vid_priv
    except FileNotFoundError:
        print("El directorio de contenidos no existe")

    #  Configuracion basica del socket
    dir_ip_servidor = "127.0.0.1"
    puerto_servidor = 8888
    dir_socket_servidor =(dir_ip_servidor, puerto_servidor)
    s = socket(AF_INET, SOCK_STREAM)
    s.bind(dir_socket_servidor)
    s.listen()

    # Configuracion de los inputs y el select
    inputs = [s]
    clients = []
```

```
235     # Función para borrar los archivos de un directorio
236     def limpiar_directorio(directorio):
237         for archivo in os.listdir(directorio):
238             ruta_archivo = os.path.join(directorio, archivo)
239             os.remove(ruta_archivo)
240             print(f"\nArchivo eliminado: {ruta_archivo}")
241         print("Carpeta",directorio,"limpia")
242         return None
243 
244     # Función para enviar mensajes
245     def enviar_msg(msg, destino):
246         try:
247             destino.sendall(msg.encode())
248         except:
249             destino.sendall(msg)
250         print("\nMensaje enviado")
251         return None
252 
253     # Función para enviar archivos
254     def enviar_archivo(filename,destino):
255         with open(filename,"r") as file:
256             data = file.read()
257         enviar_msg(data,destino)
258         print("\nArchivo",filename,"enviado")
259         return None
```

```
261     # Funcion aviso tipo de archivo y encriptado
262     def aviso_archivo_enviar(destino,tipo_archivo,encriptado):
263         mensaje = ""
264         if encriptado:
265             mensaje = "ENCRYPTED:"
266         mensaje += tipo_archivo
267         enviar_msg(mensaje,destino)
268         print("\nAviso de tipo de archivo enviado")
269         return None
270 
271     def listar_archivos():
272         """
273         Crea una lista con los archivos de un directorio.
274 
275         Returns:
276             lista_archivos_str (String): Cadena de texto con la lista
277             que contiene los nombres de los archivos disponibles para enviar
278         """
279         if not(contenido):
280             return "No hay contenido disponible"
281         else:
282             lista_archivos_str = "Contenido Disponible:\n" + "\n".join(contenido)
283             print(lista_archivos_str)
284             return lista_archivos_str
```

```
286     # Manejo del servidor
287
288     while inputs:
289         print("\nEsperando comandos...")
290         r2r, r2w, ex = select.select(inputs,[],[])
291         for msg in r2r:
292             if msg == s:
293                 cliente, dir_cliente = msg.accept()
294                 print("\nSe conectó al servidor",dir_cliente)
295                 inputs.append(cliente)
296                 clients.append(cliente)
297             else:
298                 datos = msg.recv(2048).decode()
299                 print(datos)
300                 if datos:
301                     if datos == "list":
302                         lista_contenido = listar_archivos()
303                         enviar_msg(lista_contenido,msg)
304
305                     elif datos == "fin":
306                         print("\n",msg,"se desconectó")
307                         inputs.remove(msg)
308                         clients.remove(msg)
309                         msg.close()
310                         exit(0)
```

```
312                 elif datos.startswith("enviar"):
313                     archivo = datos[6:].replace(" ","")
314
315                     if archivo in contenido:
316                         if archivo in img_public:
317                             ruta = "contenido/publico/imagenes/"+archivo
318                             tipo = "IMG"
319                             encrypted = False
320
321                         elif archivo in vid_public:
322                             ruta = "contenido/publico/videos/"+archivo
323                             tipo = "VID"
324                             encrypted = False
325
326                         elif archivo in img_priv:
327                             ruta = "contenido/cifrado/imagenes_cifradas/"+archivo
328                             tipo = "IMG"
329                             encrypted = True
330
331                         else:
332                             ruta = "contenido/cifrado/videos_cifrados/"+archivo
333                             tipo = "VID"
334                             encrypted = True
```

```
336                             salida = "enviar/"+archivo
337
338                             if tipo == "IMG":
339                                 id = msg.getpeername()
340                                 marcaAgua_visible(ruta,salida,str(id))
341                                 ruta = salida
342                                 salida = salida[:-3]+"txt"
343                                 imagen2texto(ruta,salida)
344
345                             if tipo == "VID":
346                                 salida = salida[:-3]+"txt"
347                                 video2texto(ruta,salida)
348
349                             if encrypted:
350                                 resultado = encriptar_archivo(salida,salida)
351                                 longitud= str(len(resultado)).encode()
352
353                             aviso_archivo_enviar(msg,tipo,encrypted)
354
355                             try:
356                                 enviar_msg(longitud,msg)
357                                 enviar_msg(resultado,msg)
358                             except:
359                                 enviar_archivo(salida,msg)
360
361                             limpiar_directorio("enviar")
362
363                         else:
364                             enviar_msg("ERROR",msg)
365
```

```
365
366                     else:
367                         respuesta_comando_incorrecto = "ERROR"
368                         enviar_msg(respuesta_comando_incorrecto,msg)
369
370                 else:
371                     exit(0)
372
373     return None
374
375 ##################################################################
376 #         Funcionamiento básico del programa
377 ##################################################################
378
379 if __name__ == "__main__":
380     servidor()
```

Servidor de licencias (code):

```python
import os
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.primitives import serialization, padding
import socket
import json
import ast

# FUNCIONES -----------------------------------------------------------
# Función para generar la clave e IV para cifrar la comunicación
def generar_claves():
    # Generar una clave AES (16 bytes/128 bits) y un IV (16 bytes)
    clave = os.urandom(16)                                          # Genera una clave de 16 bytes
    iv = os.urandom(16)                                             # Genera un IV de 16 bytes
    return clave, iv                                                # Devuelve los valores en formato hexadecimal

# Función para cifrar con AES en modo CBC
def cifrar_contenido(contenido, key, iv):
    """
    Función para cifrar el contenido utilizando AES en modo CBC.
    Importante: La variable contenido debe ser en binario
    """
    # Crear un cifrador AES con modo CBC usando el IV proporcionado
    cipher = Cipher(algorithms.AES(key), modes.CBC(iv))
    encryptor = cipher.encryptor()

    # Agregar padding al contenido
    padder = padding.PKCS7(128).padder()
    contenido_padded = padder.update(contenido) + padder.finalize()

    # Cifrar el contenido
    contenido_cifrado = encryptor.update(contenido_padded) + encryptor.finalize()

    return contenido_cifrado

# Función para descifrar el contenido en caso de que esté cifrado.
def descifrar_contenido(contenido_encriptado, key, iv):
    """
    Función para descifrar el contenido en caso de que esté cifrado.
    Importante: El variable contenido debe ser en binario
    """
    # Crear un cifrador AES con modo CBC usando el IV recibido
    cipher = Cipher(algorithms.AES(key), modes.CBC(iv))
    decryptor = cipher.decryptor()

    # Desencriptar el contenido
    contenido_descifrado = decryptor.update(contenido_encriptado) + decryptor.finalize()

    # Eliminar el padding
    unpadder = padding.PKCS7(128).unpadder()
    contenido_descifrado = unpadder.update(contenido_descifrado) + unpadder.finalize()

    return contenido_descifrado


# Función para cifrar en RSA con pow
def cifrar_rsa_bytes(mensaje, e, n):
    """
    Cifra un mensaje en bytes usando la clave pública RSA.
    """
    # Convertir cada byte del mensaje en un entero y cifrarlo
    mensaje_cifrado = [pow(byte, e, n) for byte in mensaje]
    return mensaje_cifrado

# Función para comprobar la autenticidad de una firma
def comprobar_firma(mensaje_firmado,e,n):
    """
    Comprueba la autenticidad de la firma digital de 'x'
    Devuelve el mensaje original

    Args:
    mensaje_firmado: firma digital
    e: clave pública
    n: clave pública
    """

    return "".join([chr(pow(byte, e, n)) for byte in mensaje_firmado])

```

```python
# Función para cifrar las claves e IVs de los contenidos
def cifrar_json(input_json, output_json, clave, iv):
    with open(input_json, "r") as archivo:                              # Abrir el JSON original
        datos = json.load(archivo)                                      # Cargar datos (claves e IV)

    # Diccionario para almacenar los datos cifrados
    datos_cifrados = {}

    for contenido, valores in datos.items():
        #Acceder al valor de "clave" y "iv" (en bytes)
        clave_original = valores["clave"].encode()
        iv_original = valores["iv"].encode()


        # Cifrar la clave e IV
        clave_cifrada = cifrar_contenido(clave_original,clave, iv)
        iv_cifrado = cifrar_contenido(iv_original,clave, iv)

        # Almacenar los datos (formato hexadecimal) cifrados en el nuevo diccionario
        datos_cifrados[contenido] = {
            "clave_cifrada": clave_cifrada.hex(),
            "iv_cifrado": iv_cifrado.hex()
        }

    # Guardar el JSON cifrado en el archivo de salida
    with open(output_json, "w") as archivo_salida:
        json.dump(datos_cifrados, archivo_salida, indent=4)             # Indent 4 para estructura mas legible

    print(f"Archivo JSON cifrado guardado en: {output_json}")

 # Generamos la clave e IV para cifrar el JSON
clave_maestra, iv_maestro = generar_claves()

cifrar_json("claves_contenido.json", "claves_contenido_cifradas.json", clave_maestra, iv_maestro)


# COMUNICACIÓN TCP-----------------------------------------------------------

# Host y puerto de la comunicación
```

```python
# COMUNICACIÓN TCP-----------------------------------------------------------

# Host y puerto de la comunicación
host = '127.0.0.1'                                                      # Host (debe coincidir con el de la app)
port = 8889                                                             # Puerto (debe coincidir con el de la app)

# Función para enviar clave e IV para descifrar comunicación
def des_comunicacion():
     # Clave e IV utilizados para cifrar la clave e IV de los contenidos
    global clave_maestra, iv_maestro

    # Crear socket del servidor
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind((host, port))
    server_socket.listen(1)

    print(f"Esperando conexión en {host} : {port}")

    # Aceptar la conexión
    client_socket, client_address = server_socket.accept()
    print(f"Conexión establecida con {client_address}")

    # Recibir clave pública enviada por la aplicación (separa en e y n)
    e_app, n_app = client_socket.recv(10000).decode().split(",")

    # Generar nuevas claves que cifran la comunicación
    clave_com, iv_com = generar_claves()

    # Cifrar clave e IV para enviarlas al cliente
    clave_com_cif = str(cifrar_rsa_bytes(clave_com, int(e_app), int(n_app)))
    iv_com_cif = str(cifrar_rsa_bytes(iv_com, int(e_app), int(n_app)))

    # Enviar claves cifradas
    client_socket.sendall(clave_com_cif.encode())
    client_socket.sendall(iv_com_cif.encode())

    # Llamar a la función para recibir y procesar los datos
    procesar_datos(client_socket, clave_com, iv_com)

```

```
    # Cerrar los sockets
    client_socket.close()
    server_socket.close()

# Función para procesar los datos recibidos del cliente
def procesar_datos(client_socket, clave_com, iv_com):
    # Firma digital y contenido
    contenido_cif = client_socket.recv(100000000)
    contenido = descifrar_contenido(contenido_cif, clave_com, iv_com).decode()
    contenido, firma = contenido.split(":")
    contenido_mod = "enviar/"+contenido[:-3]+"txt"
#     print("CONTENIDOMOD",contenido_mod)
    firma = ast.literal_eval(firma)

    # Clave pública para verificar la firma
    k_pu_cif = client_socket.recv(10000)
    k_pu = descifrar_contenido(k_pu_cif, clave_com, iv_com).decode()
    e_firma, n_firma = k_pu.split(",")
    e_firma = int(e_firma)
    n_firma = int(n_firma)

    # Verificar la firma
    firma_check = comprobar_firma(firma, e_firma, n_firma)

    if not firma_check:
        print("Firma incorrecta")
    else:
        print("Solicitud recibida:", contenido)

        with open("claves_contenido_cifradas.json", "r") as archivo:
            claves_contenido = json.load(archivo)

        # Buscar la solicitud para obtener el contenido solicitado
        if contenido_mod in claves_contenido:
            # Buscar la clave e IV asociados al contenido
            clave_contenido_cifrado = claves_contenido[contenido_mod]["clave_cifrada"]
            iv_contenido_cifrado = claves_contenido[contenido_mod]["iv_cifrado"]

            # Descifrar la clave e IV asociados al contenido
            clave_contenido_descifrado = descifrar_contenido(bytes.fromhex(clave_contenido_cifrado), clave_maestra, iv_maestro)
            iv_contenido_descifrado = descifrar_contenido(bytes.fromhex(iv_contenido_cifrado), clave_maestra, iv_maestro)

#             print(clave_contenido_descifrado)

            # Cifrar la clave e IV con el cifrado de comunicación
            clave_cont = cifrar_contenido(clave_contenido_descifrado, clave_com, iv_com)
            iv_cont = cifrar_contenido(iv_contenido_descifrado, clave_com, iv_com)
#             print("Clave contenido:", clave_contenido_descifrado)
#             print("IV contenido:", iv_contenido_descifrado)

            # Enviar la clave e IV de los contenidos de manera cifrada
            client_socket.sendall(clave_cont)
            client_socket.sendall(iv_cont)
            print("Licencia enviada.")
        else:
            print("No se encontró ese archivo")

# Función principal
def main():
    des_comunicacion()

# Comprobamos si el archivo es ejecutado directamente
if __name__ == "__main__":
    main()
```

UA (User Application) (code):

```
import socket
import ast
import base64
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.primitives import padding,serialization
from cryptography.hazmat.primitives.asymmetric import rsa


# Configuración del servidor de contenidos
servidor_contenidos_ip = '127.0.0.1'
servidor_contenidos_puerto = 8888

# Configuración del servidor de licencias
servidor_licencias_ip = '127.0.0.1'
servidor_licencias_puerto = 8889

# Configuración del CDM
cdm_ip = '127.0.0.1'
cdm_puerto = 8887
cdm_puerto = 8887

############################################################
#           Funciones básicas del programa
############################################################

def verificar_cifrado(contenido):
    """
    Función para verificar si el contenido está cifrado.
    """
    # Comprobar si el contenido tiene un encabezado específico,
    return contenido.startswith("ENCRYPTED:")

def recibir_contenido():
    """
    Función para recibir contenido del servidor de contenidos.
    """

    cliente = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

            # Recibir la lista de contenidos disponibles
            lista_contenidos = cliente.recv(10240).decode()
            print("\n")
            print(lista_contenidos)

        elif archivo=="fin":
            cliente.sendall(b"fin")
            cliente.close()
            exit(0)

        else:
            # Enviar la solicitud para descargar el archivo seleccionado
            cliente.sendall(b"enviar " + archivo.encode())

            # Recibir tipo del contenido (y si está cifrado)
            tipo = cliente.recv(10240).decode()

            if tipo != 'ERROR':  # Si se recibe contenido válido, proceder
```

```
            es_cifrado = verificar_cifrado(tipo)

            if es_cifrado:
                # Recibir tamaño del contenido
                size_data = cliente.recv(10240).decode()  # Recibir tamaño como string
                contenido_tamano = int(size_data)  # Convertir a entero

                # Recibir el contenido completo en fragmentos
                contenido = b""
                recibido = 0
                while recibido < contenido_tamano:
                    fragmento = cliente.recv(10240)  # Recibir en bloques
                    print("Fragmento:",fragmento)
                    if not fragmento:
                        break
                    contenido += fragmento
                    recibido += len(fragmento)

                # Verificar si se recibió todo
                # Verificar si se recibió todo
                if recibido == contenido_tamano:
                    print(f"Contenido recibido completamente ({recibido} bytes).")
                else:
                    print(f"Error: Se esperaba {contenido_tamano} bytes pero se recibieron {recibido} b

            else:
                contenido = cliente.recv(10240000).decode()

            # Procesar contenido según si está cifrado
            if es_cifrado:
                return True, contenido, tipo[len("ENCRYPTED:"):], archivo
            else:
                return False, contenido, tipo, archivo

        else:
            print("No se pudo descargar el archivo. Intenta nuevamente.")

    return None

def recibir_clave_RSA(ip_destino,puerto_destino):
    """
    Envía la clave pública al receptor y recibe la clave e IV cifrados.
    Descifra la clave e IV utilizando la clave privada RSA.
    """

    # e,n: clave pública
    #   d: clave privada
    e,n,d = generar_claves_rsa()

    cliente.connect((ip_destino, puerto_destino))

    if (puerto_destino == 8887):
        print("Conectado al CDM.")
    if (puerto_destino == 8888):
```

```
    if (puerto_destino == 8888):
        print("Conectado al servidor de contenidos.")
    if (puerto_destino == 8889):
        print("Conectado al servidor de licencias.")

    # Enviar los componentes de la clave pública
    clave_publica_serializada = "{0},{1}".format(e,n)
#     print("CLAVE RSA ENVIADA:",clave_publica_serializada)
    cliente.sendall(clave_publica_serializada.encode())

    # Recibir la clave e IV para comunicacion (cifrada con RSA)
    clave_comunicacion_cifrada = cliente.recv(10240).decode()
#     print(clave_comunicacion_cifrada)
    iv_comunicacion_cifrado = cliente.recv(10240).decode()
#     print(iv_comunicacion_cifrado)

    # Conversión de string "[1,2,3]" a lista [1,2,3]
    clave_comunicacion_cifrada= ast.literal_eval(clave_comunicacion_cifrada)
    iv_comunicacion_cifrado= ast.literal_eval(iv_comunicacion_cifrado)

    # Descifrar la clave e IV usando la clave privada
    clave_comunicacion = descifrar_rsa_bytes(clave_comunicacion_cifrada, d, n)
    iv_comunicacion = descifrar_rsa_bytes(iv_comunicacion_cifrado, d, n)

    return clave_comunicacion,iv_comunicacion

def pedir_licencia_CDM(licencia_deseada):
    """
    Función para pedir la solicitud de licencia al cdm.
    """
    cdm_key, cdm_iv = recibir_clave_RSA(cdm_ip,cdm_puerto)

    # Pedir solicitud para licencia del contenido deseado
    cliente.sendall(encriptar_texto(licencia_deseada.encode(), cdm_key, cdm_iv)) # Mensaje en bytes

    # Recibir la solicitud de licencia
    solicitud_licencia_enc = cliente.recv(102400000)
    # Recibir la solicitud de licencia
    solicitud_licencia_enc = cliente.recv(102400000)
    solicitud_licencia = desencriptar_texto(solicitud_licencia_enc, cdm_key, cdm_iv).decode()
#     print(solicitud_licencia)

    # Recibir la clave pública del CDM
    k_pu_cdm_enc = cliente.recv(10240)
    k_pu_cdm = desencriptar_texto(k_pu_cdm_enc, cdm_key, cdm_iv).decode()
    print(f"Solicitud de licencia para '{licencia_deseada}' adquirida.")

    return solicitud_licencia,k_pu_cdm


def pedir_licencia_servidor(solicitud_licencia,clave_publica_cdm):
    """
    Función para pedir la licencia al servidor de licencias usando la solicitud del CDM.
    """
    licencias_key, licencias_iv = recibir_clave_RSA(servidor_licencias_ip,servidor_licencias_puerto)
```

```
#     print(solicitud_licencia)

    # Solicitud de licencia al servidor
    cliente.sendall(encriptar_texto(solicitud_licencia.encode(), licencias_key, licencias_iv)) # Mensaje en
    cliente.sendall(encriptar_texto(clave_publica_cdm.encode(), licencias_key, licencias_iv)) # Mensaje en

    # Recibir la licencia
    clave_contenido_enc = cliente.recv(10240)
    clave_contenido = desencriptar_texto(clave_contenido_enc, licencias_key, licencias_iv).decode()
    iv_contenido_enc = cliente.recv(10240)
    iv_contenido = desencriptar_texto(iv_contenido_enc, licencias_key, licencias_iv).decode()
    print(f"Clave adquirida del servidor de licencias.")

    licencia = "{0},{1}".format(clave_contenido,iv_contenido)
#     print(licencia)

    return licencia


def descifrar_contenido_CDM(licencia,contenido_cifrado):
    """
    Función para pedir al CDM que descifre el contenido con la licencia proporcionada.
    """
    cdm_key, cdm_iv = recibir_clave_RSA(cdm_ip,cdm_puerto)

    # Enviar licencia al CDM
    cliente.sendall(encriptar_texto(licencia.encode(), cdm_key, cdm_iv)) # Mensaje en bytes

    # Enviar la longitud del contenido
    longitud= str(len(contenido_cifrado)).encode()

    cliente.sendall(encriptar_texto(longitud, cdm_key, cdm_iv))

    # Enviar contenido cifrado
    cliente.sendall(encriptar_texto(contenido_cifrado, cdm_key, cdm_iv)) # Mensaje en bytes

    # Recibir contenido descifrado

    # Recibir contenido descifrado

    # Recibir tamaño del contenido
    size_data_enc = cliente.recv(10240)  # Recibir tamaño como string
    size_data= desencriptar_texto(size_data_enc, cdm_key, cdm_iv).decode()
    contenido_tamano = int(size_data)  # Convertir a entero

    # Recibir el contenido completo en fragmentos
    contenido_enc = b""
    recibido = 0
    while recibido < contenido_tamano:
        fragmento = cliente.recv(10240)  # Recibir en bloques
        if not fragmento:
            break
        contenido_enc += fragmento
        recibido += len(fragmento)

    # Verificar si se recibió todo
```

```
    # Verificar si se recibió todo
    if recibido == contenido_tamano:
        print(f"Contenido recibido completamente ({recibido} bytes).")
    else:
        print(f"Error: Se esperaba {contenido_tamano} bytes pero se recibieron {recibido} bytes.")

    contenido_descifrado = desencriptar_texto(contenido_enc, cdm_key, cdm_iv)
    print(f"Contenido descifrado por CDM con éxito.")

    return contenido_descifrado


############################################################
#         Funciones relacionados con cifrado RSA
############################################################

def descifrar_rsa_bytes(mensaje_cifrado, d, n):
    """
def descifrar_rsa_bytes(mensaje_cifrado, d, n):
    """
    Descifra un mensaje cifrado usando la clave privada RSA.
    """
    # Descifrar cada entero del mensaje cifrado y convertirlo a bytes
    mensaje_descifrado = bytes([pow(byte, d, n) for byte in mensaje_cifrado])
    return mensaje_descifrado

def generar_claves_rsa():
    """
    Genera un par de claves RSA usando la librería cryptography.
    Retorna la clave pública y privada, junto con los componentes (e, n).
    """
    # Generar clave privada
    clave_privada = rsa.generate_private_key(
        public_exponent=65537,
        key_size=2048
    )
    # Obtener clave pública
    # Obtener clave pública
    clave_publica = clave_privada.public_key()

    # Extraer componentes de la clave pública
    numeros_publicos = clave_publica.public_numbers()
    e = numeros_publicos.e
    n = numeros_publicos.n

    numeros_privados = clave_privada.private_numbers()
    d = numeros_privados.d

    return (e, n, d)


############################################################
#      Funciones relacionados con cifrado AES en modo CBC
############################################################

def encriptar_texto(texto, clave, iv):
```

```
def encriptar_texto(texto, clave, iv):
    """
    Cifra un texto utilizando AES en modo CBC con relleno PKCS7.

    Args:
        texto (bytes): El texto a cifrar.
        clave (bytes): La clave de cifrado (16 bytes).
        iv (bytes): El vector de inicialización (16 bytes).

    Returns:
        bytes: El texto cifrado.
    """
    cipher = Cipher(algorithms.AES(clave), modes.CBC(iv))
    encryptor = cipher.encryptor()

    padder = padding.PKCS7(algorithms.AES.block_size).padder()
    padded_data = padder.update(texto) + padder.finalize()

    return encryptor.update(padded_data)
def desencriptar_texto(texto_cifrado, clave, iv):
    """
    Descifra un texto cifrado utilizando AES en modo CBC y elimina el relleno PKCS7.

    Args:
        texto_cifrado (bytes): El texto cifrado.
        clave (bytes): La clave de cifrado (16 bytes).
        iv (bytes): El vector de inicialización (16 bytes).

    Returns:
        bytes: El texto descifrado sin relleno.
    """
    cipher = Cipher(algorithms.AES(clave), modes.CBC(iv))
    decryptor = cipher.decryptor()

    decipher_result = decryptor.update(texto_cifrado)

    unpadder = padding.PKCS7(algorithms.AES.block_size).unpadder()
    return unpadder.update(decipher_result) + unpadder.finalize()


############################################################
#         Funciones para transformar contenidos
############################################################

# Convertir un string de contenido a imagen
def string_to_image(image_string, output_path):
    image_data = base64.b64decode(image_string.encode("utf-8"))
    with open(output_path, "wb") as image_file:
        image_file.write(image_data)

# Convertir un string de contenido a video
def string_to_video(video_string, output_path):
    video_data = base64.b64decode(video_string.encode("utf-8"))
    with open(output_path, "wb") as video_file:
        video_file.write(video_data)
```

```
############################################################
#     FASE 1: Comunicación con servidor de contenidos
############################################################

ide,contenido,tipo,archivo = recibir_contenido()


############################################################
#     FASE 2: Comunicación con CDM
############################################################

# Verificar si el contenido está cifrado
# En cuyo caso, pedir al CDM una solicitud de licencia para el servidor de licencias
if ide:
    # Socket de la UA
    cliente = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    solicitud_licencia, clave_publica_cdm = pedir_licencia_CDM(archivo)
    cliente.close()

    ############################################################
    #     FASE 3: Comunicación con servidor de licencias
    ############################################################

    # Socket de la UA
    cliente = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # Conexión con el servidor de licencias
    licencia = pedir_licencia_servidor(solicitud_licencia,clave_publica_cdm)
    cliente.close()


    ############################################################
    #     FASE 4: Descifrado del contenido desde CDM
    ############################################################

    # Socket de la UA
    cliente = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # Conexión con el CDM para obtener el contenido descifrado
    contenido_descifrado = descifrar_contenido_CDM(licencia,contenido).decode()
    print(f"'{archivo}' descifrado y almacenado con éxito en el equipo.")
    cliente.close()

    # Guardar contenido
    if tipo=="IMG":
        string_to_image(contenido_descifrado, "descarga/"+archivo)
    else:
        string_to_video(contenido_descifrado, "descarga/"+archivo)

else:
    # Guardar contenido
    if tipo=="IMG":
        string_to_image(contenido, "descarga/"+archivo)
```

CDM (Content Decryption Module) (code):

```python
import socket
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.primitives import padding,serialization
from cryptography.hazmat.primitives.asymmetric import rsa
import os

############################################################
#           Funciones básicas del programa
############################################################

def generar_claves():
    """
    Genera la clave e IV para cifrar la comunicación
    """
    # Generar una clave AES (16 bytes/128 bits) y un IV (16 bytes)
    clave = os.urandom(16)                                      # Genera una clave de 16 bytes
    iv = os.urandom(16)                                         # Genera un IV de 16 bytes
    return clave, iv                                            # Devuelve los valores en formato hexadecimal

def enviar_clave_RSA():
    """
    Recibe la clave pública del emisor y envía la clave e IV cifrados.
    """
    # Clave pública
    e_ua, n_ua = ua_socket.recv(10240).decode().split(",")

    # Clave e IV cifrados con la clave pública RSA
    clave_cif = str(cifrar_rsa_bytes(clave, int(e_ua), int(n_ua)))
    iv_cif    = str(cifrar_rsa_bytes(iv,    int(e_ua), int(n_ua)))

    # Enviar clave e iv cifrados a UA
    ua_socket.sendall(clave_cif.encode())
    ua_socket.sendall(iv_cif.encode())
    print("Clave e IV enviados a UA.")

def recibir_peticion_ua():
    # Enviar clave e IV para establecer conexión segura
    enviar_clave_RSA()

    peticion_ua_enc = ua_socket.recv(10240)
    peticion_ua = desencriptar_texto(peticion_ua_enc, clave, iv).decode()
    print(f"Petición de UA recibida. Licencia solicitada: {peticion_ua}")

    return peticion_ua
```

```python
def enviar_solicitud_licencia_firmada(peticion_ua):
    # Generar clave privada para firmar digitalmente
    # Y clave pública para enviar al servidor para que compruebe la firma
    e,n,d = generar_claves_rsa()

    # Crear solicitud y firmar digitalmente
    x = peticion_ua
    s = firmar_msg(x,d,n)
    s = "{0}".format(s)

    solicitud_firmada = "{0}:{1}".format(x,s).encode()
    k_pu = "{0},{1}".format(e,n).encode()

    # Enviar a UA
    ua_socket.sendall(encriptar_texto(solicitud_firmada, clave, iv)) # Mensaje en bytes
    ua_socket.sendall(encriptar_texto(k_pu, clave, iv)) # Mensaje en bytes
    print(f"Enviada solicitud de licencia firmada digitalmente para '{x}'")
```

```python
def firmar_msg(mensaje,d,n):
    """
    Firma digitalmente el mensaje 'x' y devuelve

    Args:
    > mensaje: mensaje a firmar (str/byte)
    > d: clave privada
    > n: clave pública
    """

    # El mensaje tiene que ser tipo 'bytes'
    if not isinstance(mensaje, bytes):
        mensaje = mensaje.encode()

    return [pow(byte, d, n) for byte in mensaje]
```

```python
def descifrar_contenido():
    """
    Recibe contenido y licencia asociada y desencripta
    contenido y envía a UA.
    """

    # Enviar clave e IV para establecer conexión segura
    enviar_clave_RSA()

    # Recibir licencia para desencriptar contenido
    licencia_enc = ua_socket.recv(10240)
    licencia = desencriptar_texto(licencia_enc, clave, iv).decode()
    clave_contenido, iv_contenido = licencia.split(",")
    clave_contenido = bytes.fromhex(clave_contenido)
    iv_contenido = bytes.fromhex(iv_contenido)

    # Recibir contenido cifrado

    # Recibir tamaño del contenido
    size_data_enc = ua_socket.recv(10240)
    size_data= desencriptar_texto(size_data_enc, clave, iv).decode()
    # Recibir tamaño como string
    contenido_tamano = int(size_data)  # Convertir a entero

    # Recibir el contenido completo en fragmentos
    contenido_cifrado_enc = b""
    recibido = 0
    while recibido < contenido_tamano:
        fragmento = ua_socket.recv(10240)  # Recibir en bloques
        if not fragmento:
            break
        contenido_cifrado_enc += fragmento
        recibido += len(fragmento)

    # Verificar si se recibió todo
    if recibido == contenido_tamano:
        print(f"Contenido recibido completamente ({recibido} bytes).")
    else:
        print(f"Error: Se esperaba {contenido_tamano} bytes pero se recibieron {recibido} bytes.")

    contenido_cifrado = desencriptar_texto(contenido_cifrado_enc, clave, iv)

    # Desencriptar contenido con licencia
    contenido_txt = desencriptar_texto(contenido_cifrado, clave_contenido, iv_contenido)
    print(f"Contenido descifrado con éxito con el uso de licencia")
```

```python
    # Enviar a UA

    # Enviar la longitud del contenido
    longitud= str(len(contenido_txt)).encode()
    ua_socket.sendall(encriptar_texto(longitud, clave, iv))

    ua_socket.sendall(encriptar_texto(contenido_txt, clave, iv)) # Mensaje en bytes
```

```python
############################################################
#         Funciones relacionados con cifrado RSA
############################################################

def cifrar_rsa_bytes(mensaje, e, n):
    """
    Cifra un mensaje en bytes usando la clave pública RSA.
    """
    # Convertir cada byte del mensaje en un entero y cifrarlo
    mensaje_cifrado = [pow(byte, e, n) for byte in mensaje]
    return mensaje_cifrado

def generar_claves_rsa():
    """
    Genera un par de claves RSA usando la librería cryptography.
    Retorna la clave pública y privada, junto con los componentes (e, n).
    """
    from cryptography.hazmat.primitives.asymmetric import rsa

    # Generar clave privada
    clave_privada = rsa.generate_private_key(
        public_exponent=65537,
        key_size=2048
    )
    # Obtener clave pública
    clave_publica = clave_privada.public_key()

    # Extraer componentes de la clave pública
    numeros_publicas = clave_publica.public_numbers()
    e = numeros_publicas.e
    n = numeros_publicas.n

    numeros_privadas = clave_privada.private_numbers()
    d = numeros_privadas.d

    return (e, n, d)


```

```python
############################################################
#      Funciones relacionados con cifrado AES en modo CBC
############################################################

def encriptar_texto(texto, clave, iv):
    """
    Cifra un texto utilizando AES en modo CBC con relleno PKCS7.

    Args:
        texto (bytes): El texto a cifrar.
        clave (bytes): La clave de cifrado (16 bytes).
        iv (bytes): El vector de inicialización (16 bytes).

    Returns:
        bytes: El texto cifrado.
    """
    cipher = Cipher(algorithms.AES(clave), modes.CBC(iv))
    encryptor = cipher.encryptor()

    padder = padding.PKCS7(algorithms.AES.block_size).padder()
    padded_data = padder.update(texto) + padder.finalize()

    return encryptor.update(padded_data)

```

```python
def desencriptar_texto(texto_cifrado, clave, iv):
    """
    Descifra un texto cifrado utilizando AES en modo CBC y elimina el relleno PKCS7.

    Args:
        texto_cifrado (bytes): El texto cifrado.
        clave (bytes): La clave de cifrado (16 bytes).
        iv (bytes): El vector de inicialización (16 bytes).

    Returns:
        bytes: El texto descifrado sin relleno.
    """
    cipher = Cipher(algorithms.AES(clave), modes.CBC(iv))
    decryptor = cipher.decryptor()

    decipher_result = decryptor.update(texto_cifrado)

    unpadder = padding.PKCS7(algorithms.AES.block_size).unpadder()
    return unpadder.update(decipher_result) + unpadder.finalize()


```

```python
################################################################################
#     FASE 2 (general): Comunicación UA-CDM (solicitud de licencia)
################################################################################

# Configuración del CDM
cdm_ip = '127.0.0.1'
cdm_puerto = 8887

# Configuración para recibir conexión de UA
cdm_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
cdm_socket.bind((cdm_ip, cdm_puerto))
cdm_socket.listen(1)
print(f"Esperando conexión en {cdm_ip} : {cdm_puerto}")

# Socket de UA
ua_socket, ua_address = cdm_socket.accept()
print(f"Conexión establecida con {ua_address}")

# Clave e IV para cifrado AES en modo CBC
clave, iv = generar_claves()

peticion_ua = recibir_peticion_ua()
enviar_solicitud_licencia_firmada(peticion_ua)
cdm_socket.close()

```

```python
################################################################################
#     FASE 4 (general): Descifrado del contenido desde CDM
################################################################################

# Configuración para recibir conexión de UA
cdm_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
cdm_socket.bind((cdm_ip, cdm_puerto))
cdm_socket.listen(1)

# Socket de UA
ua_socket, ua_address = cdm_socket.accept()
print(f"Conexión establecida con {ua_address}")

# Desencriptación del contenido con licencia y envío a UA
descifrar_contenido()

# Cerrar el socket del CDM
cdm_socket.close()
```