SEGURIDAD Y DERECHOS DIGITALES

PROYECTO FINAL

PABLO FERNÁNDEZ BEAUS

JORGE VALERO RÓDENAS

GONZALO BARONA TRÉNOR

DAVID RAMOS DOMINGO

GRUPO A-3

ÍNDICE

INTRODUCCION.

En el contexto actual, la salvaguarda de contenidos digitales se ha vuelto crucial. Este proyecto, se centra en la creación de un sistema robusto de protección, abordando los principios fundamentales compartidos por sistemas DRM como FairPlay, Widevine y PlayReady.

En este proyecto, nos enfocaremos en la programación en Python de elementos clave: una aplicación, un servidor de contenidos y un servidor de licencias. El objetivo principal es comprender y aplicar eficientemente los principios de funcionamiento de los sistemas DRM existentes.

El proyecto se divide en dos fases esenciales. La primera implica la interacción de la aplicación con el servidor de contenidos para adquirir recursos digitales. La segunda fase se adentra en la complejidad de la protección digital, con la aplicación enfrentándose a contenidos cifrados y buscando claves de descifrado gestionadas por el servidor de licencias.

En este proyecto la implementación no se limita a lo básico. Se debe incorporar marcas de agua en imágenes y firmas digitales en mensajes de solicitud de claves para una mayor seguridad.

En busca de la excelencia, la máxima calificación se logra mediante la segmentación de la aplicación de usuario en dos programas: una aplicación de usuario (UA) y un Content Decryption Module (CDM). Este enfoque especializado mejora la modularidad y eficiencia del sistema.

Este proyecto explora la criptografía, la seguridad de las comunicaciones y la gestión de derechos digitales, representando nuestro compromiso con la vanguardia de la tecnología digital. Bienvenidos a la era de la seguridad digital, donde nuestro sistema se erige como guardián de la integridad y los derechos en el mundo digital

DESCRIPCIÓN.

En este apartado, detallaremos paso a paso el funcionamiento del proyecto. Lo primero de todo, hay que aclarar que en este proyecto no se ha usado la biblioteca **`cryptography`** para generar las claves RSA y AES CTR. Para generar las claves RSA hemos utilizado la función *generar_claves()* que se encarga de buscar números de 16 cifras primos para calcular (n,e),(n,d).

Para generar las claves AES CTR utilizamos claves que son ya strings, la única diferencia es que las generamos nosotros aleatoriamente y tenemos que enviarlas con encode() y recibirlas con decode(). Aquí la comparamos con OS:

Comencemos conectando el servidor principal, el servidor de claves y el CDM para que esperen una conexión. El servidor principal alberga 3 imágenes y dos videos, cifrados con AES en modo CTR, así como versiones sin cifrar. Al conectar la UA (cliente), esta se conecta inicialmente al servidor principal y espera uno de los dos comandos: 'LIST' o 'GET'. Con 'LIST ALL', se muestra el listado de nombres de los archivos en el servidor, permitiendo búsquedas más específicas como .jpg o .mp4. Por otro lado, el comando 'GET' inicia la descarga del archivo especificado.

A partir de este punto, se inicia un proceso lleno de solicitudes y cifrados. El servidor principal lee el archivo solicitado y verifica si es una imagen o no. En caso de ser una imagen, se verifica si está cifrada. En caso afirmativo, se establece conexión con la función *conectar_claves_servidorRSA(),* la cual implica un intercambio de claves públicas entre el servidor principal y el servidor de claves para comunicarse mediante cifrado RSA. El servidor principal envía el nombre del archivo cifrado con RSA y un certificado digital al servidor de claves. Este último valida el certificado y, si es correcto, genera una clave de cifrado y un vector de inicialización (IV) para el cifrado en modo CTR, enviándolos al servidor principal mediante RSA. El servidor principal recibe y descifra la clave y el IV mediante RSA.

El servidor de claves utiliza la función *cargar_claves_desde_archivo()* para descifrar un archivo de texto que almacena las claves de descifrado de los archivos. Extrae las claves del archivo solicitado, las encripta con la clave e IV creadas previamente y las envía al servidor principal. Este último recibe las claves cifradas en modo CTR, las descifra con la clave e IV recibidas anteriormente y desencripta la imagen solicitada por la UA.

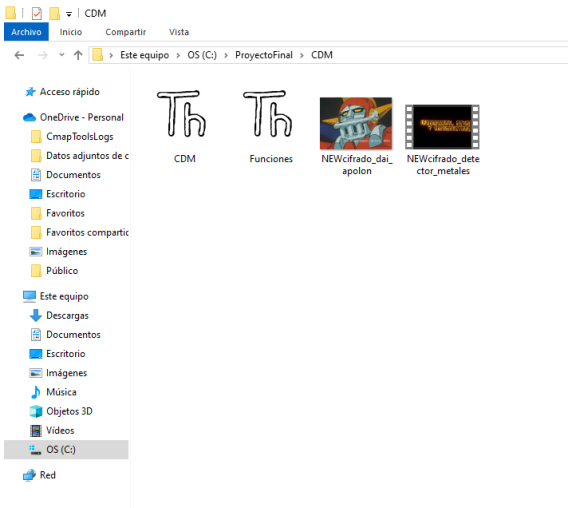
Posteriormente, el servidor principal emplea la función *agregar_marca_agua()* para crear una copia de la imagen solicitada con una marca de agua visible, que contiene la IP y el puerto de la UA. Luego, cifra nuevamente los archivos en modo CTR y envía la imagen cifrada con la marca de agua a la UA. Si la imagen no está cifrada, se le añade una marca de agua y se envía directamente a la UA. En caso de ser un video, se repite el proceso, pero sin agregar una marca de agua.

Cuando la UA recibe la descarga de un archivo, verifica si está cifrado. En caso afirmativo, se conecta con el CDM mediante la función *enviar_descarga(),* que envía el archivo como se describió anteriormente.

La UA y el CDM inician una conversación RSA similar a la descrita anteriormente, con comunicación simétrica y asimétrica. La diferencia radica en que el CDM prepara una petición para el servidor de claves utilizando la función *preparar_peticion(),* que consiste en enviar tres valores, el nombre del archivo, la firma digital de este, y la clave pública para descifrar la firma digital. Esta petición se cifra en modo CTR y se envía a la UA, quien la transmite al servidor de claves mediante la función *conectar_servidor_clavesRSA().*

El servidor de claves recibe e identifica la petición, comprueba si es una petición mediante los espacios. Separa las variables, la descifra y verifica la firma digital. Si es válida, sigue con el protocolo estándar enviando las claves a la UA. La UA recibe las claves, las envía cifradas al CDM, y finalmente, el CDM las recibe y desencripta con éxito el archivo cifrado, almacenándolo en su directorio.

Si los archivos no están cifrados, la UA los recibe directamente del servidor principal en su directorio, marcando el fin de este proceso.



CÓDIGO.

Servidor Principal:

```
from socket import*
from Funciones import*
import os
import time
from PIL import Image, ImageDraw, ImageFont
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms,
modes

host = 'localhost' # 127.0.0.1
port = 60000  # Puerto del Servidor principal
portKeys = 61000  # Puerto del servidor de licencias

def agregar_marca_agua(ima, usuario_id):
    # Cargar la imagen
    imagen = Image.open(ima)
    # Crear un objeto ImageDraw para dibujar en la imagen
    draw = ImageDraw.Draw(imagen)
    # Configurar la fuente y el tamaño para la marca de agua
    font = ImageFont.load_default()
    # Configurar el contenido de la marca de agua (puedes personalizar
esto según tus necesidades)
    marca_de_agua = f"Marca de agua, Usuario:{usuario_id}"
    # Obtener las dimensiones de la imagen
    width, height = imagen.size
    # Calcular la posición para la marca de agua (puedes personalizar la
posición)
    x = width - 265
    y = height - 50
    # Agregar la marca de agua a la imagen
    draw.text((x, y), marca_de_agua, font=font, fill=(255, 255, 255,
128))
    # Guardar la imagen con la marca de agua
    imagen_con_marca = f"con_marca{ima}"
    imagen.save(imagen_con_marca)

    return imagen_con_marca

def LIST(e):
    lista = os.listdir();lis=[]
    for i in lista:
        if i.endswith(str(e[-3:])):
            lis.append(i)
    return lis
```

```
def descarga(n, conexion, usuario_id):
    try:
        nombre = n[4:]
        archivo_read = open(nombre,"rb")
        ArchivoEncode = archivo_read.read()

        if nombre.endswith(('jpg','jpeg','png','bmp')):

            if verificar_marcador(nombre):
                print("Archivo cifrado, solicitando claves")
                # Petición de claves al servidor de licencias via RSA
                clave,iv = conectar_claves_servidorRSA(nombre)
                # Desencriptar archivo con las claves
recibidas.
                desencriptar_archivo(nombre, clave.encode(), iv.encode())
                agregar_marca_agua(nombre, usuario_id)
                encriptar_archivo("con_marca"+nombre,"con_marca"+nombre,
clave.encode(), iv.encode())
                encriptar_archivo(nombre, nombre, clave.encode(),
iv.encode())

                read_con_marca = open("con_marca"+nombre,"rb")
                tamaño_marca = os.path.getsize("con_marca"+nombre)
                #Enviar tamaño del archivo
                conexion.sendall(str(tamaño_marca).encode())
                time.sleep(0.01)
                Archivo_con_marca = read_con_marca.read()
                conexion.sendall(Archivo_con_marca)
                print(f"Enviado {len(Archivo_con_marca)} bytes")
                # Cerrar y eliminar el archivo con la marca de agua
después de enviarlo
                read_con_marca.close()
                os.remove("con_marca" + nombre)
            else:
                agregar_marca_agua(nombre, usuario_id)
                read_con_marca = open("con_marca"+nombre,"rb")
                tamaño_marca = os.path.getsize("con_marca"+nombre)
                conexion.sendall(str(tamaño_marca).encode())
                time.sleep(0.01)
                Archivo_con_marca = read_con_marca.read()
                conexion.sendall(Archivo_con_marca)
                print(f"Enviado {len(Archivo_con_marca)} bytes")
                # Cerrar y eliminar el archivo con la marca de agua
después de enviarlo
                read_con_marca.close()
                os.remove("con_marca" + nombre)
        else:
            tamaño_total = os.path.getsize(nombre)
            conexion.sendall(str(tamaño_total).encode())
```

```
            time.sleep(0.01)
            conexion.sendall(ArchivoEncode)
            print(f"Enviado {len(ArchivoEncode)} bytes")
    except FileNotFoundError:
        conexion.sendall("401 FICHERO NO ENCONTRADO.".encode())
    finally:
        archivo_read.close()

def tamaño(p):
    size = os.path.getsize(p[4:])
    return size

serversocket = socket(AF_INET, SOCK_STREAM)
TCPserver = ('localhost', 60000)
print("conectando en {}, en el puerto {}".format(*TCPserver))
serversocket.bind(TCPserver)

serversocket.listen(5)

while True:
    sock,client_address = serversocket.accept()
    print(f"Nueva conexión establecida...{client_address}")
    while True:
        listado = os.listdir();listado_conv= tuple(listado)
        peticion = sock.recv(1024)
        start= peticion.decode();concreto = LIST(start)

        if (start.startswith("LIST") == True) or (start.startswith("GET")
== True) or (start == "QUIT"):
            if (start.startswith("LIST") == True):
                if start.endswith(" ALL"):
                    sock.send(("200 INICIO DE ENVIO
LISTADO..."+"\n").encode())
                    sock.send(str(listado_conv).encode())

                elif concreto == []:
                    sock.send("201 NO HAY FICHEROS.".encode())
                else:
                    sock.send(("200 INICIO DE ENVIO
LISTADO..."+"\n").encode())
                    sock.send(str(concreto).encode())

            elif (start.startswith("GET") == True):
                try:
                    ok=("202 LONGITUD DEL
CONTENIDO:"+str(tamaño(start))+" bytes"+"\n")
                    sock.send(ok.encode())
                    descarga(start,sock,client_address)
                except:
```

```
                    sock.send("401 FICHERO NO
ENCONTRADO.".encode())

            elif (start.startswith("QUIT") == True):
                sock.send("Gracias por usar este programa.".encode())
                sock.close()

        else:
            nook=("400 ERROR. MENSAJE NO IDENTIFICADO"+"\n")
            sock.send(nook.encode())

        print(f"{client_address} dice: ",start)
#sock.close()
```

Servidor de claves:

```
from socket import*
from Funciones import*
import time
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms,
modes
import os

# Función para encriptar un archivo en modo CTR
def encriptar_archivo(archivo_entrada, archivo_salida, key, iv):
    cifrado = Cipher(algorithms.AES(key), modes.CTR(iv))
    encriptado = cifrado.encryptor()

    with open(archivo_entrada, 'rb') as entrada, open(archivo_salida,
'wb') as salida:

        for bloque in entrada:
            bloque_encriptado = encriptado.update(bloque)
            salida.write(bloque_encriptado)

# Función que maneja el encriptado de una cadena de texto en modo CTR,
sin marcador.
def encriptar_texto(texto, key, iv):
    # Convertir la cadena de texto a bytes
    texto_bytes = texto.encode('utf-8')

    # Crear el objeto Cipher
    cifrado = Cipher(algorithms.AES(key), modes.CTR(iv))
    encriptado = cifrado.encryptor()

    # Encriptar la cadena de texto
    texto_encriptado = encriptado.update(texto_bytes)
```

```
    # Devolver el texto encriptado como bytes
    return texto_encriptado

# Función que maneja el desencriptado en modo CTR de un archivo
def desencriptar_archivo(archivo_entrada, key, iv):
    cifrado = Cipher(algorithms.AES(key), modes.CTR(iv))
    desencriptado = cifrado.decryptor()

    with open(archivo_entrada, 'rb') as entrada:
        contenido_desencriptado = b''
        for bloque in entrada:
            bloque_desencriptado = desencriptado.update(bloque)
            contenido_desencriptado += bloque_desencriptado
    return contenido_desencriptado

# Función que desencripta modo CTR el archivo .txt y extrae las claves
del archivo
def cargar_claves_desde_archivo(ruta_archivo,key,iv):
    claves = []
    desencriptado = desencriptar_archivo('licencias_cifradas.txt', key,
iv).decode('latin-1')

    for linea in desencriptado.split('\n'):
        partes = linea.strip().split(';')
        nombre_archivo = partes[0].strip()
        clave = partes[1].strip()
        vi = partes[2].strip()
        if nombre_archivo == ruta_archivo:
            claves += clave, vi
    return ";".join(map(str, claves))

# Claves con la que está cifrado modo CTR el archivo .txt con las
licencias de los archivos.
key = b'c'*16
iv = b'c'*16
#encriptar_archivo('licencias.txt', 'licencias_cifradas.txt', key, iv)

# Conexión TCP
server_socket = socket(AF_INET, SOCK_STREAM)
server_socket.bind(('localhost', 61000))
server_socket.listen(5)

while True:
    print("Servidor esperando conexiones...")
    # Esperamos conexiones
    client_socket, client_address = server_socket.accept()
    print(f"Conexión establecida con {client_address}")
    # Generamos clave pública y privada. Envia clave publica al cliente
```

```
    clave_publica, clave_privada = generar_claves()
    client_socket.send(str(clave_publica).encode())
    time.sleep(0.01)
    # Recibe clave publica RSA del cliente.
    cliente_clave_publica = eval(client_socket.recv(1024).decode())
    # Recibe el nombre de archivo cifrado con RSA, desciframos con clave
privada
    recepcion = client_socket.recv(16024)
    datos_deco = recepcion.decode()
    print(datos_deco)
    descifrado = descifrar(eval(datos_deco),clave_privada)
    print(descifrado)
    time.sleep(0.01)
    if any(caracter.isspace() for caracter in descifrado):
        partes =  descifrado.rsplit('[', 1)
        print(partes)
        archivo_original = partes[0].strip()
        print(archivo_original)
        lista_numeros_y_tupla = partes[1].rsplit(']', 1)
        archivo_firma = [int(numero) for numero in
lista_numeros_y_tupla[0].split(',') if numero.strip()]
        print(archivo_firma)
        tupla = [numero for numero in
lista_numeros_y_tupla[1].strip(')').split(',')]
        publica_DCM = int(tupla[0][2:]),int(tupla[1])
        print(publica_DCM)
        verificar_firma = descifrar(archivo_firma,publica_DCM)
        print(verificar_firma)
        if archivo_original == verificar_firma:
            print("Firma digital confirmada")
            print(f'Petición claves de: {client_address}:\n{descifrado}')
            auxiliar = client_socket.recv(1024)
            time.sleep(0.01)
            # Genera claves aleatorias (clave e IV) para modo CTR
            numero_clave_cifrado = ''.join([chr(random.randint(48, 122))
for _ in range(16)])
            numero_iv_cifrado = ''.join([chr(random.randint(48, 122)) for
_ in range(16)])
            # Cifra modo RSA con clave pública del cliente y lo envía
            cifrar_numero_clave = cifrar(numero_clave_cifrado,
cliente_clave_publica)
            cifrar_numero_iv = cifrar(numero_iv_cifrado,
cliente_clave_publica)
            client_socket.send(str(cifrar_numero_clave).encode())
            time.sleep(0.01)
            client_socket.send(str(cifrar_numero_iv).encode())
            time.sleep(0.01)
            # Saca las claves del archivo pedido de un archivo .txt
encriptado
```

```
            claves = cargar_claves_desde_archivo(archivo_original,key,iv)
            # Encripta esas claves modoo CTR con la clave e IV generadas
            encriptar_claves =
encriptar_texto(claves,numero_clave_cifrado.encode(),numero_iv_cifrado.en
code())
            client_socket.send(encriptar_claves)
            print('Claves enviadas con exito')
            #client_socket.close()
        else:
            client_socket.send("Firma digital no confirmada").encode()
            client_socket.close()

    else:
        #Recepcion de firma digital
        recepcion_firma = client_socket.recv(1024)
        datos_deco_firma = recepcion_firma.decode()
        descifrado_firma =
descifrar(eval(datos_deco_firma),cliente_clave_publica)
        print(f'Petición claves de: {client_address}:\n{descifrado}')
        time.sleep(0.01)
        if descifrado == descifrado_firma:
            print("Firma digital confirmada")
            print(f'Petición claves de: {client_address}:\n{descifrado}')
            # Genera claves aleatorias (clave e IV) para modo CTR
            numero_clave_cifrado = ''.join([chr(random.randint(48, 122))
for _ in range(16)])
            numero_iv_cifrado = ''.join([chr(random.randint(48, 122)) for
_ in range(16)])
            # Cifra modo RSA con clave pública del cliente y lo envía
            cifrar_numero_clave = cifrar(numero_clave_cifrado,
cliente_clave_publica)
            cifrar_numero_iv = cifrar(numero_iv_cifrado,
cliente_clave_publica)
            client_socket.send(str(cifrar_numero_clave).encode())
            time.sleep(0.01)
            client_socket.send(str(cifrar_numero_iv).encode())
            time.sleep(0.01)
            # Saca las claves del archivo pedido de un archivo .txt
encriptado
            claves = cargar_claves_desde_archivo(descifrado,key,iv)
            # Encripta esas claves modoo CTR con la clave e IV generadas
            encriptar_claves =
encriptar_texto(claves,numero_clave_cifrado.encode(),numero_iv_cifrado.en
code())
            client_socket.send(encriptar_claves)
            print('Claves enviadas con exito')
            #client_socket.close()
        else:
            client_socket.send("Firma digital no confirmada").encode()
```

```
            client_socket.close()

#client_socket.close()
```

CDM:

```
import socket
import os
import time
from Funciones import*
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms,
modes

host = 'localhost' # 127.0.0.1
port = 60000  # Puerto del Servidor principal
portKeys = 61000  # Puerto del servidor de licencias

def conectar_UA(client_socket):
    publica,privada = generar_claves()
    client_socket.send(str(publica).encode())
    clave_publica_UA = client_socket.recv(1024).decode()
    time.sleep(0.01)
    # Recibimos la clave y el IV para desencriptar.
    respuesta_encriptada_clave = client_socket.recv(1024).decode()
    time.sleep(0.01)
    respuesta_encriptada_iv = client_socket.recv(1024).decode()
    time.sleep(0.01)
    # Desciframos con RSA (clave privada), la clave y el IV
    clave = descifrar(eval(respuesta_encriptada_clave), privada)
    iv = descifrar(eval(respuesta_encriptada_iv), privada)
    print(clave,iv)
    archivo_cifrado = client_socket.recv(1024).decode()
    archivo =
desencriptar_texto(eval(archivo_cifrado),clave.encode(),iv.encode())
    print(archivo)
    solicitud = preparar_peticion(archivo)
    solicitud_cifrada =
encriptar_texto(solicitud,clave.encode(),iv.encode())
    client_socket.send(str(solicitud_cifrada).encode())
    time.sleep(0.01)
    claves = client_socket.recv(1024).decode()
    time.sleep(0.01)
    claveMain, ivMain = claves.split()
    #client_socket.close()
    return claveMain, ivMain

def preparar_peticion(archivo):
    clave_publica,clave_privada = generar_claves()
```

```
    firma_digital = cifrar(archivo, clave_privada)
    peticion = f"{archivo} {firma_digital} {clave_publica}"
    return peticion

def recibir_descarga(socket):
    # Nombre del nuevo archivo
    nombre = socket.recv(1026).decode()
    time.sleep(0.01)
    descarga = open(nombre,"wb")
    size_data = socket.recv(1026)
    time.sleep(0.01)
    # Recibimos el tamaño del archivo desde el servidor
    tam = int(size_data.decode())
    print(f"Datos recibidos: {tam}")
    cont=True
    rec = 0
    # Comenzamos la descarga del archivo desde el servidor
    while cont:
        data = socket.recv(1026)
        rec += len(data)
        descarga.write(data)
        # Cuando el coincida con el tamaño el bucle para
        if rec == tam:
            cont=False
    descarga.close()

    tamaño_total = os.path.getsize(nombre)
    # Verificamos que el tamaño recibido coincide con el que tenemos
    print(f"Total recibidos: {tamaño_total}")
    clave,iv = conectar_UA(socket)
    # Desencriptar archivo con las claves recibidas.
    desencriptar_archivo(nombre, clave.encode(), iv.encode())

DCMsocket = socket(AF_INET, SOCK_STREAM)
DCM_sock = ('localhost', 62000)
print("conectando en {}, en el puerto {}".format(*DCM_sock))
DCMsocket.bind(DCM_sock)

DCMsocket.listen(5)

while True:
    print("Servidor esperando conexiones...")
    # Esperamos conexiones
    client_socket, client_address = DCMsocket.accept()
    print(f"Conexión establecida con {client_address}")
    recibir_descarga(client_socket)
  client_socket.close()
```

UA (Cliente):

```
import socket
import os
import time
from Funciones import*
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms,
modes


host = 'localhost' # 127.0.0.1
port = 60000  # Puerto del Servidor principal
portKeys = 61000  # Puerto del servidor de licencias
portDCM = 62000  # Puerto del DCM

#socket TCP
client_socket = socket(AF_INET, SOCK_STREAM)

client_socket.connect((host, port))
print(f"Conectado al servidor en {host}:{port}")

# Función para desencriptar un archivo en modo CTR
def desencriptar_archivo(archivo_entrada, key, iv):
    cifrado = Cipher(algorithms.AES(key), modes.CTR(iv))
    desencriptado = cifrado.decryptor()
    with open(archivo_entrada, 'rb') as entrada:
        contenido_cifrado = entrada.read()
        contenido_cifrado = contenido_cifrado[:-7]
        contenido_desencriptado = desencriptado.update(contenido_cifrado)

    with open(archivo_entrada, 'wb') as sobrescribir:
        sobrescribir.write(contenido_desencriptado)

# Función para desencriptar na cadena de texto en modo CTR
def desencriptar_texto(texto_encriptado, key, iv):
    # Crear el objeto Cipher
    cifrado = Cipher(algorithms.AES(key), modes.CTR(iv))
    desencriptado = cifrado.decryptor()

    # Desencriptar el texto
    texto_desencriptado_bytes = desencriptado.update(texto_encriptado)

    # Convertir los bytes desencriptados a una cadena de texto
    texto_desencriptado = texto_desencriptado_bytes.decode('utf-8')

    # Devolver la cadena de texto desencriptada
    return texto_desencriptado
```

```
# Función para manejar la recepción de mensajes que no tengan que ver con
'LIST' O 'GET'
def recibir_mensajes():
    while True:
        try:
            data = client_socket.recv(1024)
            if data:
                print(f"\nMensaje recibido: {data.decode('utf-8')}")
                break
            else:
                print("El servidor ha cerrado la conexión.")
                break
        except Exception as e:
            print(f"Error al recibir mensajes: {e}")
            break

def conectar_DCM(archivo,client_socket_DCM):

    # Recibimos clave publica RSA del servidor de licencias
    DCM_clave_publica = eval(client_socket_DCM.recv(1024).decode())

    # Generar clave publica y privada
    clave_publica, clave_privada = generar_claves()
    client_socket_DCM.send(str(clave_publica).encode())
    time.sleep(0.01)
    # Genera claves aleatorias (clave e IV) para modo CTR
''.join([chr(random.randint(48, 122)) for _ in range(16)])
    numero_clave_cifrado = ''.join([chr(random.randint(48, 122)) for _ in
range(16)])
    numero_iv_cifrado = ''.join([chr(random.randint(48, 122)) for _ in
range(16)])
    # Cifra modo RSA con clave pública del cliente y lo envía
    cifrar_numero_clave = cifrar(numero_clave_cifrado, DCM_clave_publica)
    cifrar_numero_iv = cifrar(numero_iv_cifrado, DCM_clave_publica)
    client_socket_DCM.send(str(cifrar_numero_clave).encode())
    time.sleep(0.01)
    client_socket_DCM.send(str(cifrar_numero_iv).encode())
    time.sleep(0.01)

    archivo_cifrado =
encriptar_texto(archivo[3:],numero_clave_cifrado.encode(),numero_iv_cifra
do.encode())
    client_socket_DCM.send(str(archivo_cifrado).encode())
    time.sleep(0.01)

    peticion = client_socket_DCM.recv(2024).decode()
    time.sleep(0.01)
```

```
    peticion_desencriptada =
desencriptar_texto(eval(peticion),numero_clave_cifrado.encode(),numero_iv
_cifrado.encode())

    clave, iv = conectar_claves_servidorRSA(peticion_desencriptada)
    claves = f"{clave} {iv}"

    client_socket_DCM.send(str(claves).encode())
    time.sleep(0.01)
    #client_socket_DCM.close()


def enviar_descarga(nombre):
    client_socket_DCM = socket(AF_INET, SOCK_STREAM)
    client_socket_DCM.connect((host, portDCM))
    archivo_read = open(nombre,"rb")
    ArchivoEncode = archivo_read.read()
    tamaño_total = os.path.getsize(nombre)
    client_socket_DCM.sendall(str(nombre).encode())
    time.sleep(0.01)
    client_socket_DCM.sendall(str(tamaño_total).encode())
    time.sleep(0.01)
    client_socket_DCM.sendall(ArchivoEncode)
    print(f"Enviado a DCM {len(ArchivoEncode)} bytes")
    #client_socket_DCM.close()
    conectar_DCM(nombre,client_socket_DCM)
    #client_socket_DCM.close()

# Función para manejar la recepción de la descarga
def recibir_descarga(mensaje,socket):
    # Nombre del nuevo archivo
    nombre = "NEW"+mensaje[4:]
    descarga = open(nombre,"wb")
    size_data = socket.recv(1026)
    # Recibimos el tamaño del archivo desde el servidor
    tam = int(size_data.decode())
    print(f"Datos recibidos: {tam}")
    cont=True
    rec = 0
    # Comenzamos la descarga del archivo desde el servidor
    while cont:
        data = socket.recv(1026)
        rec += len(data)
        descarga.write(data)
        # Cuando el coincida con el tamaño el bucle para
        if rec == tam:
            cont=False
    descarga.close()
```

```
    tamaño_total = os.path.getsize(nombre)
    # Verificamos que el tamaño recibido coincide con el que tenemos
    print(f"Total recibidos: {tamaño_total}")
    # Comprobamos si está cifrado o no
    if verificar_marcador(nombre):
        print("Archivo cifrado")
        enviar_descarga(nombre)

    else:
        print("Archivo no cifrado")

# Bucle principal para enviar mensajes
while True:
    message = input("Ingrese un mensaje ('QUIT' para salir, 'GET' para
descargar, 'LIST ALL' para listado ): ")
    client_socket.sendall(message.encode())
    peticion =  client_socket.recv(1024);peti_deco = peticion.decode()
    if peti_deco.startswith('202'):
        try:
            recibir_descarga(message,client_socket)
        except Exception as e:
            print(f"Error al recibir el archivo: {e}")

    elif message.upper() == 'QUIT':
        break

    elif peti_deco.startswith('200'):
        recibir_mensajes()

    else:
        print("El servidor denegó tu petición.\n")


# Cerrar el socket
#client_socket.close()
```

Funciones Adicionales:

```
import random
import math
import time
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms,
modes
from socket import*

host = 'localhost' # 127.0.0.1
port = 60000  # Puerto del Servidor principal
portKeys = 61000  # Puerto del servidor de licencias

def generar_claves():
    #Genera dos númeos primos aleatorios
    p = generar_primo()
    q = generar_primo()

    #Calcular el módulo n y la función de Euler phi
    n = p * q
    phi = (p - 1) * (q -1)

    #Elegir una clave pública y que sea coprima con phi
    e = clave_publica(phi)

    #Calcular la clave privada
    d = modinv(e,phi)

    return ((n,e),(n,d))

def generar_primo():
    #Generar un número primo aleatorio
    while True:
        num = random.randint(2**25,2**26-1)
        if primo(num):
            return num

def primo(numero):
    #Verificar si es un número primo
    for i in range(2, int(math.sqrt(numero))+1):
        if numero % i == 0:
            return False
    return True

def clave_publica(phi):
    #Elegir una clave publica y que sea coprima con phi
    while True:
        e = random.randint(2,phi-1)
        if math.gcd(e,phi) == 1:
```

```
            return e

def modinv(a,m):
    #Calcular el inverso modular usando el algoritmo extendido de
Euclides
    m0, x0, x1 = m,0, 1
    while a > 1:
        q = a // m
        m, a = a % m, m
        x0, x1 = x1 - q * x0, x0
    return x1 + m0 if x1 < 0 else x1

def cifrar(mensaje, clave_publica):
    # Cifrar el mensaje usando la clave publica (n, e)
    n, e = clave_publica
    # Convertir el texto a una lista de números representando los
caracteres
    mensaje_numerico = [ord(char) for char in mensaje]
    # Cifrar cada número por separado
    mensaje_cifrado_numerico = [pow(char, e, n) for char in
mensaje_numerico]
    return mensaje_cifrado_numerico

def descifrar(mensaje_cifrado, clave_privada):
    # Descifrar un mensaje cifrado usando la clave privada (n, d)
    n, d = clave_privada
    # Descifrar cada número por separado
    mensaje_descifrado_numerico = [pow(char, d, n) for char in
mensaje_cifrado]
    # Convertir los números a caracteres y unirlos en un solo texto
    mensaje_descifrado_texto = ''.join([chr(char) for char in
mensaje_descifrado_numerico])
    return mensaje_descifrado_texto

# Función para encriptar un archivo en modo CTR
def encriptar_archivo(archivo_entrada, archivo_salida, key, iv):
    cifrado = Cipher(algorithms.AES(key), modes.CTR(iv))
    encriptado = cifrado.encryptor()

    with open(archivo_entrada, 'rb') as entrada:
        bloque_encriptado = entrada.read()
        bloque_encriptado = encriptado.update(bloque_encriptado)
    with open(archivo_salida, 'wb') as salida:
        salida.write(bloque_encriptado)
        salida.write(b'cifrado')

# Función para desencriptar un archivo en modo CTR
def desencriptar_archivo(archivo_entrada, key, iv):
    cifrado = Cipher(algorithms.AES(key), modes.CTR(iv))
```

```
    desencriptado = cifrado.decryptor()
    with open(archivo_entrada, 'rb') as entrada:
        contenido_cifrado = entrada.read()
        contenido_cifrado = contenido_cifrado[:-7]
        contenido_desencriptado = desencriptado.update(contenido_cifrado)

    with open(archivo_entrada, 'wb') as sobrescribir:
        sobrescribir.write(contenido_desencriptado)

# Función que maneja el encriptado de una cadena de texto en modo CTR,
sin marcador.
def encriptar_texto(texto, key, iv):
    # Convertir la cadena de texto a bytes
    texto_bytes = texto.encode('utf-8')

    # Crear el objeto Cipher
    cifrado = Cipher(algorithms.AES(key), modes.CTR(iv))
    encriptado = cifrado.encryptor()

    # Encriptar la cadena de texto
    texto_encriptado = encriptado.update(texto_bytes)

    # Devolver el texto encriptado como bytes
    return texto_encriptado

# Función para desencriptar na cadena de texto en modo CTR
def desencriptar_texto(texto_encriptado, key, iv):
    # Crear el objeto Cipher
    cifrado = Cipher(algorithms.AES(key), modes.CTR(iv))
    desencriptado = cifrado.decryptor()

    # Desencriptar el texto
    texto_desencriptado_bytes = desencriptado.update(texto_encriptado)

    # Convertir los bytes desencriptados a una cadena de texto
    texto_desencriptado = texto_desencriptado_bytes.decode('utf-8')

    # Devolver la cadena de texto desencriptada
    return texto_desencriptado

# Función que conecta con el servidor de licencias via RSA
def conectar_claves_servidorRSA(archivo):
    client_socket_licencia = socket(AF_INET, SOCK_STREAM)
    client_socket_licencia.connect((host, portKeys))
    # Recibimos clave publica RSA del servidor de licencias
    servidor_clave_publica =
eval(client_socket_licencia.recv(1024).decode())
    print(servidor_clave_publica)
    # Generar clave publica y privada
```

```
    clave_publica, clave_privada = generar_claves()
    client_socket_licencia.send(str(clave_publica).encode())
    time.sleep(0.01)

    # Cifrar el nombre de archivo con la clave pública del servidor
    cifrar_archivo = cifrar(archivo, servidor_clave_publica)
    client_socket_licencia.sendall(str(cifrar_archivo).encode())
    time.sleep(0.01)
    #Firma digital
    firmar_archivo = cifrar(archivo,clave_privada)
    client_socket_licencia.sendall(str(firmar_archivo).encode())
    time.sleep(0.01)
    print(firmar_archivo)
    # Recibimos la clave y el IV para desencriptar.
    respuesta_encriptada_clave =
client_socket_licencia.recv(1024).decode()
    time.sleep(0.01)
    respuesta_encriptada_iv = client_socket_licencia.recv(1024).decode()
    time.sleep(0.01)
    # Desciframos con RSA (clave privada), la clave y el IV
    clave_descifrada = descifrar(eval(respuesta_encriptada_clave),
clave_privada)
    iv_descifrada = descifrar(eval(respuesta_encriptada_iv),
clave_privada)
    print(clave_descifrada,iv_descifrada)
    data = client_socket_licencia.recv(1024)
    print('Claves recibidas del servidor de claves.\nDesencriptando
archivo...')
    # Desencriptamos las claves que nos mandan en modo CTR
    lista =
desencriptar_texto(data,clave_descifrada.encode(),iv_descifrada.encode())
    lista = lista.split(";")
    print("¡Archivo desencriptado con exito!")
    #client_socket_licencia.close()
    return lista
# Función para verificar si un archivo esta encriptado o no
def verificar_marcador(archivo):
    with open(archivo, 'rb') as entrada:
        entrada.seek(-7, 2)
        marcador = entrada.read()
        return marcador == b'cifrado'
```

CUADERNO DE BITÁCORA, DIAGRAMA GANT Y AUTOEVAULACIÓN.

Nombres	Horas empleadas
David Ramos Domingo	40 horas
Pablo Fernández Beaus	20 horas
Jorge Valero Ródenas	20 horas
Gonzalo Barona Trénor	20 horas

DIAGRAMA DE GANT

David Ramos

Jorge Valero

AUTOEVAULACIÓN:

Nombre	Autoevaulación	Motivos
David Ramos	9	Algún fallo no deseado en el código
Pablo Fernandez	9.5	Gran trabajo en general
Gonzalo Barona	8	Podría haber dedicado más tiempo
Jorge Valero	7	Conocimientos justos pero buena actitud de cara al proyecto.

CONCLUSIONES.

A lo largo de la realización del proyecto, hemos llegado a las siguientes conclusiones:

Hemos obtenido valiosos conocimientos acerca de los sistemas DRM más utilizados en la actualidad, como FairPlay, Widevine y PlayReady. Lo cual ha sido de vital importancia para el desarrollo del proyecto.

La implementación del proyecto nos ha permitido adquirir ciertas habilidades en criptografía simétrica y asimétrica, así como en la aplicación de firmas digitales. El proyecto también nos ha ayudado a comprender en mayor profundidad los algoritmos de cifrado vistos en teoría y prácticas como AES o RSA.

No todo salió a pedir de boca desde el principio, ya que nos encontramos con algunos problemillas técnicos y de seguridad, pero nada que no pudiera ser solucionado. Tuvimos que asegurarnos de que la comunicación entre las partes estuviera bien encriptada, lidiar con las claves de cifrado y poner nuestra firma digital en los mensajes. Fue un trabajo difícil, pero finalmente dimos con las soluciones necesarias.

Este proyecto nos hizo ver lo fascinante que puede ser el mundo de la seguridad y la criptografía. Enfrentar desafíos y trabajar en la encriptación de la comunicación nos mostró la importancia y la emoción de este campo.

En resumen, el proyecto ha sido una oportunidad valiosa para aplicar conocimientos teóricos a situaciones prácticas, desarrollando habilidades técnicas y estratégicas esenciales en el campo de la seguridad digital y la gestión de derechos en entornos digitales

´

BIBLIOGRAFÍA:

UPV: https://poliformat.upv.es/portal/site/GRA_14212_2023/tool/21e8b8ea-adba-42ce-83d4-2fc360f2dd88