PROYECTO SEDGER

Aiden Bugarín Carreira, Adrian Martínez Quivén, Ángela Fernández Fernández y Paula Bosch Ramos

ÍNDICE

1.- INTRODUCCIÓN

Este proyecto se enfoca en el desarrollo de un sistema de protección de contenidos digitales basado en los principios fundamentales de los sistemas DRM (Digital Rights Management). El objetivo es brindar a los estudiantes una comprensión práctica de cómo funcionan estos sistemas y cómo se pueden implementar soluciones de protección de contenido digital.

El sistema consta de una aplicación de usuario, un servidor de contenidos y un servidor de licencias, y se exploran funcionalidades adicionales como marcas de agua y firmas digitales. A lo largo de esta memoria, se detalla la implementación y los resultados obtenidos, lo que contribuirá al desarrollo de habilidades técnicas en seguridad digital.

2.- DESARROLLO

Podremos decir que el trabajo se separa en tres niveles de especificaciones:

1. En el primer nivel se encuentra la realización básica de la comunicación del cliente con los dos servidores. En el primer paso vemos que el cliente puede pedirle al servidor de contenidos alguna imagen, y este se la envía, pero puede estar cifrado o no estarlo. Una vez el cliente tiene el contenido averigua si está o no cifrado, si no está cifrado simplemente descarga y puede ver la imagen, pero si está cifrado empieza la comunicación con el servidor de licencias. Este le facilita la clave de cifrado simétrico que se ha empleado para cifrar la imágen, todo esto mediante conversación entre cliente y servidor de licencias de manera cifrada, para que no hayan ataques de por medio. Una vez el cliente o aplicación disponga de la clave de descifrado, obtendrá el contenido de imagen sin cifrar y podrá visualizar la imágen.

2. En esta segunda fase incluimos las marcas de agua para marcar los contenidos que queramos. Si se trata de una imagen, el servidor de contenidos incluirá una marca de agua visible cuyo contenido dependerá del usuario al que le envíe la información y servirá para poder identificarlo. El mensaje de solicitud de la clave de cifrado deberá estar firmado digitalmente por la aplicación, para que todo sea más seguro.

3. Finalmente, el último nivel del trabajo consistiría en separar el cliente o aplicación. Separando el cliente tenemos la UA (aplicación de usuario) y CDM. Cuando la UA obtiene un fichero cifrado le pide al CDM el mensaje de solicitud de licencias, y el CDM se encarga de preparar dicho mensaje y firmarlo

digitalmente. Después, se lo manda de nuevo a la UA, quien lo reenvía al servidor de licencias. El servidor de licencias, tras comprobar la firma de mensajes, envía a la UA la clave de cifrado del contenido. Por último, la UA envía la clave al CDM y este desencripta la imagen o vídeo. Además, cabe mencionar que todas las comunicaciones entre CDM y UA estarán cifradas.

3.- RESULTADOS

A continuación incluímos los códigos fuente de cada uno de nuestros programas. Podremos ver comentarios a lo largo del código explicando la función de cada parte.

3.1. Servidor de contenidos

```
##############################
#                            #
#          SERVIDOR          #
#                            #
##############################

#LIBRERIAS---------------------------------------------------------------------
import os
from socket import *
from PIL import Image, ImageDraw, ImageFont

#FUNCIONES---------------------------------------------------------------------
from funciones import cifra_CTR

#CONFIGURACION SOCKETS---------------------------------------------------------
serverPort = 60000
serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind(('', serverPort))
serverSocket.listen()

print('Server ready to receive connections')

connectionSocket, clientAdress = serverSocket.accept()

cifrado = " "

#NOMBRE USUARIO----------------------------------------------------------------
nombre_usuario = connectionSocket.recv(2048).decode()

#PROGRAMA----------------------------------------------------------------------
while True:
    try:
        #Recibe e imprime el comando
        message = connectionSocket.recv(2048).decode()
        print("\n***Comando recibido:",message.lower())
        lista = message.split()
        #Comando LIST--------------------
```

```
#Enumera los ficheros disponibles
if lista[0] == 'LIST':
    #comprueba si se ha pedido una extension
    if lista[1] == 'ALL':
        extension = ''
    else:
        extension = '.' + lista[1].lower()

    #directorio
    directorio = 'ficheros_server/'
    contenido = os.listdir(directorio)
    fich = []

    #crea lista con los ficheros
    for fichero in contenido:
        if os.path.isfile(os.path.join(directorio, fichero)) and fichero.endswith(extension):
            fich.append(fichero)

    #si no hay ficheros
    if len(fich) == 0:
        resp = '201 NO HAY FICHEROS'
        print('\n',resp)
        connectionSocket.sendall(resp.encode())

    #hay ficheros: envia el listado
    else:
        resp = '200 INICIO ENVIO LISTADO\n'
        print(resp[:-1])

        for fichero in fich:
            resp += '\t-'+fichero+'\n'
            print('\t- ',fichero)
        connectionSocket.sendall(resp.encode())

#Comando GET-----------------------------------
#Envia un fichero
elif lista[0] == 'GET':
    try:
        extension = lista[1].split('.')[1].lower()
        nombre = lista[1].lower()
        carpeta = "ficheros_server/"

        if extension!='mp4':
            #crea fichero con marca----------------
            #Opening Image
            img = Image.open(carpeta + nombre)

            #Creating draw object
            draw = ImageDraw.Draw(img)
            #Creating text and font object
            text = nombre_usuario
            width, height = img.size
            font = ImageFont.truetype('arial.ttf', int(height/20))
            #Positioning Text
```

```
            x=int(height/40)
            y=height-int(height/20)-20

            #Applying text on image via draw object
            draw.text((x, y), text, font=font)

            #Saving the new image
            img.save("ficheros_server/temp/"+ nombre.lower())

            #cifra la imagen marcada----------
            cifra_CTR(nombre.lower())
            print("***Imagen ", nombre.lower() ,"cifrada")

            cifra_CTR("claves.txt")

            #elimina el fichero marcado sin cifrar
            os.remove("ficheros_server/temp/"+nombre.lower())

            #mandar fichero cifrado----------
            nombre = nombre.lower().split('.')
            nombre_nuevo = nombre[0]+"_e."+nombre[1]
            fichero = "ficheros_server/temp/"+nombre_nuevo

            cifrado = "T"

        else:
            fichero = "ficheros_server/"+nombre.lower()
            cifrado = "F"

        f = open(fichero,'rb')
        long = 0

        #calcula longitud del fichero
        for line in f:
            long += len(line)
        f.close()

        message = cifrado + ' 200 LONGITUD CONTENIDO: ' + str(long) + '\n'
        connectionSocket.send(message.encode())

        f2 = open(fichero,'rb')

        #manda el fichero cifrado
        for line in f2:
            connectionSocket.sendall(line)

        f2.close()
        print("---Imagen marcada y cifrada enviada") if cifrado=='T' else print("---Fichero enviado")

    except FileNotFoundError:
        message = '\n 401 FICHERO NO ENCONTRADO'
        print(message)
        connectionSocket.sendall(message.encode())
#Comando HELP----------
```

```
        elif lista[0] == 'HELP':
            print('***Imprimiendo listado de comandos')

        #Comando QUIT--------------------
        elif lista[0] == 'QUIT':
            print('\nServer has finished its work')
            break

        #Comando no reconocido-----------
        else:
            resp = '400 ERROR. MENSAJE NO IDENTIFICADO'
            print(resp)

    except Exception as e:
        print('\nERROR')
        print(e)
        connectionSocket.send("ERROR".encode())


connectionSocket.close()
serverSocket.close()
```

3.2. Servidor de licencias

```
########################################
#                                      #
#            SERVIDOR LICENCIAS        #
#                                      #
########################################

#LIBRERIAS-----------------------------------------------------------------------
import os
from socket import *
from cryptography.hazmat.primitives import serialization, hashes, padding
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.primitives.asymmetric import rsa, padding
from cryptography.hazmat.primitives.asymmetric.rsa import RSAPublicNumbers

#FUNCIONES-----------------------------------------------------------------------
from funciones import rsa_server, send, recv

#CONFIGURACION SOCKETS-----------------------------------------------------------
serverPort = 60001
serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind(('', serverPort))
serverSocket.listen()

print('Server ready to receive connections')
connectionSocket, clientAdress = serverSocket.accept()
```

```
#CONFIGURACION CIFRADO-------------------------------------------------------------
intercambio = False
KEY = os.urandom(16)
IV = os.urandom(16)

#cifrador CTR
aesCipher_CTR = Cipher(algorithms.AES(KEY),modes.CTR(IV))

#PROGRAMA--------------------------------------------------------------------------
while True:
    try:
        #Intercambio de clave simetrica----------------------
        if intercambio==False:
            rsa_server(KEY, IV, connectionSocket)
            intercambio = True

        #Comunicacion con cliente----------------------------

        #Recibe mensaje firmado por CDM y su kpub
        msj = recv(connectionSocket, KEY, IV)

        if msj == b"quit": #si el mensaje es quit --> para el programa
            break

        signature = msj.split(b"-----")[0] #recibe la firma
        print("\n***Firma recibida")

        pemCDM = recv(connectionSocket, KEY, IV) #recibe clave publica del CDM
        kpubCDM = serialization.load_pem_public_key(pemCDM)
        print("***Clave pública CDM recibida")

        #Recibe mensaje del cliente
        mensaje_b = msj.split(b"-----")[1]
        mensaje = mensaje_b.decode()
        print("***Mensaje recibido del cliente:",mensaje)

        #Verificar la firma del mensaje
        try:
            kpubCDM.verify(
                signature,
                mensaje_b,
                padding.PSS(
                    mgf=padding.MGF1(algorithm=hashes.SHA256()),
                    salt_length=padding.PSS.MAX_LENGTH,

                ),
                hashes.SHA256()
            )
            print("***Signature is valid")

        except Exception as e:
            print("ERROR\n***Signature is invalid")
            send("ERROR", connectionSocket, KEY, IV)
            break
```

```
if "." in mensaje:
    message = mensaje.split('.')[0][:-2]
else:
    message = mensaje

print("***Licencia solicitada:",message)

#Abrimos clave de claves y obtenemos las credenciales
fhand = open('temp/clave_de_claves.txt','rb')
for i,line in enumerate(fhand):
    if i==0:
        clave_de_clave = line[:16]
    if i==1:
        iv_de_clave = line[:16]
fhand.close()

#Cifrador para descifrar txt de claves
aesCipher_CTR_KEY = Cipher(algorithms.AES(clave_de_clave),modes.CTR(iv_de_clave))
aesDecryptor_CTR_KEY = aesCipher_CTR_KEY.decryptor()

#Desciframos txt de claves
fhand = open("temp/claves.txt",'rb')
data = fhand.read()
f_decrypt = aesDecryptor_CTR_KEY.update(data)+ aesDecryptor_CTR_KEY.finalize()
fhand.close()

os.remove("temp/claves.txt") #borramos el txt cifrado

f = open('temp/claves.txt', 'wb') #guardamos txt descifrado
f.write(f_decrypt)
f.close()

f = open('temp/claves.txt', 'rb')

for line in f: #buscamos clave del fichero en el txt
    nombre_ext = str(line).split("---")[0]
    nombre = nombre_ext.split(".")[0]
    l = len(nombre_ext)

    if nombre[2:] == message:
        clave = line[l+1:l+17]
        iv = line[l+20:l+36]

f.close()

if iv==b'' or clave==b'':
    send(b"ERROR", connectionSocket, KEY, IV)
else:
    msg = clave+iv
    send(msg, connectionSocket, KEY, IV)
    print("---Clave enviada:",clave)
    print("---IV enviado:",iv)
f.close()
```

```
        #Eliminamos el fichero de claves descifrado----------
        os.remove("temp/claves.txt")
        os.remove("temp/clave_de_claves.txt")

        #Eliminamos el fichero cifrado de la carpeta temporal---
        os.remove("ficheros_server/temp/"+mensaje)

    except Exception as e:
        print('\nERROR')
        print(e)
        aesEncryptor_CTR = aesCipher_CTR.encryptor()
        msg_enc = aesEncryptor_CTR.update(b"ERROR") + aesEncryptor_CTR.finalize()
        connectionSocket.send(msg_enc)

        #borra contenido de temp
        if ("claves.txt" or "clave_de_claves.txt") in os.listdir("temp/"):
            os.remove("temp/claves.txt")
            os.remove("temp/clave_de_claves.txt")

print("\nServer has finished its work")
connectionSocket.close()
serverSocket.close()
```

3.3. UA

```
#################################
#                               #
#              UA               #
#                               #
#################################

#LIBRERIAS--------------------------------------------------------------------
import os
from socket import *
from cryptography.hazmat.primitives import serialization, hashes, padding
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.primitives.asymmetric import rsa, padding
from cryptography.hazmat.primitives.asymmetric.rsa import RSAPublicNumbers, RSAPrivateNumbers

#FUNCIONES--------------------------------------------------------------------
from funciones import rsa_cliente, sign_message, send, recv

#CONFIGURACION SOCKETS--------------------------------------------------------
serverName = '127.0.0.1'
serverPort = 60000
serverPortLic = 60001
serverPortCDM = 60002

TCPserverAddr=(serverName, serverPort)
TCPserverAddrLIC=(serverName, serverPortLic)
TCPserverAddrCDM=(serverName, serverPortCDM)
```

```
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocketLIC = socket(AF_INET, SOCK_STREAM)
clientSocketCDM = socket(AF_INET, SOCK_STREAM)

clientSocket.connect(TCPserverAddr)
print("Conectado al servidor")

#CONFIGURACION CIFRADO RSA ----------------------------------------------------
intercambio_CDM = False
intercambio_LIC = False

private_key = rsa.generate_private_key(
    public_exponent=65537,
    key_size=2048,
)
public_key = private_key.public_key()

n = public_key.public_numbers().n
d = private_key.private_numbers().d

pem = public_key.public_bytes(
   encoding=serialization.Encoding.PEM,
   format=serialization.PublicFormat.SubjectPublicKeyInfo
)

firmado=False

#PROGRAMA---------------------------------------------------------------------
#Nombre de usuario-----------------------------------------
nombre_usuario = input("Por favor, introduce tu nombre: ")
clientSocket.send(nombre_usuario.encode())

while True:
    try:
        #Input y envío de comandos
        print('\nIntroduzca un comando (help para ayuda)\n')
        comando = input ('>')
        comando = comando.upper()
        comando_l = comando.split(' ')
        clientSocket.send(comando.encode())
        print('---Mensaje enviado\n')

        #Comando HELP-----------------
        #Imprime por pantalla la lista de comandos posibles
        if comando_l[0] == 'HELP':
            print("""LISTA DE COMANDOS:
            - LIST ALL: lista de todos los ficheros
            - LIST <extension>: lista los ficheros de una extensión
            - GET <fichero>: pide un fichero
            - QUIT: sale del programa
            - HELP: lista de comandos\n""")

        #Comando LIST-----------------
        #Devuelve la lista de ficheros disponibles
```

```
elif comando_l[0] == 'LIST':
    resp = clientSocket.recv(2048).decode()
    resp = resp.split('\n')

    #recorre la lista de ficheros y los imprime
    for el in resp:
        print(el)

#Comando GET--------------------
#Pide un archivo al servidor de contenidos
elif comando_l[0] == 'GET':
    resp = clientSocket.recv(2048) #respuesta inicial del servidor

    if resp == b"ERROR":
        print("ERROR \nNombre del archivo en formato incorrecto\n(recuerde poner la extension)")
        continue

    print(resp.decode()[2:])

    l_resp = resp.split(b' ')
    long2 = 0

    #Recibe el archivo
    if l_resp[1] == b'200': #200: todo correcto, se procede a enviar el archivo
        #Fichero encriptado --> añadimos _e al nombre
        if l_resp[0] == b"T": #respuesta del servidor empieza por T si el archivo esta encriptado
            print("***Fichero encriptado")
            nombre_nuevo = comando[4:].lower().split('.')
            nombre_nuevo = nombre_nuevo[0]+'_e.'+nombre_nuevo[1]
            nuevo_f = open('ficheros_recibidos/'+nombre_nuevo,'wb')

        #Fichero no encriptado --> mismo nombre
        else:
            print("***Fichero no encriptado")
            nombre_nuevo = comando[4:].lower()
            nuevo_f = open('ficheros_recibidos/'+nombre_nuevo,'wb')

        long = int(l_resp[-1].split(b':')[-1]) #longitud del fichero

        #Recibe fichero linea a linea y lo guarda
        while long2 < long:
            linea = clientSocket.recv(2048)
            long2 += len(linea)
            nuevo_f.write(linea)

        nuevo_f.close()
        nombre = nombre_nuevo.split('.')

        #Si está encriptado --> desencripta
        if nombre[0][-2:]=='_e':
            #Ciframos comunicación entre CDM y UA
            if intercambio_CDM == False:
                clientSocketCDM.connect(TCPserverAddrCDM)
                print('***Conectado al CDM')
```

```
            KEY_CDM, IV_CDM = rsa_cliente(pem, clientSocketCDM, d, n)
            intercambio_CDM = True

        #Pedimos al CDM la solicitud de licencia firmada
        send(nombre_nuevo, clientSocketCDM, KEY_CDM, IV_CDM)

        #Recibimos pem del CDM para poder verificar la firma con el serv_lic
        pemCDM = recv(clientSocketCDM, KEY_CDM, IV_CDM)
        print("***Clave pública del CDM recibida")

        #Preparamos conexión con serv_lic para enviarle solicitud de descifrar
        #Intercambio de claves
        if intercambio_LIC==False:
            clientSocketLIC.connect(TCPserverAddrLIC)
            print('***Conectado al servidor de licencias')

            KEY_LIC, IV_LIC = rsa_cliente(pem, clientSocketLIC, d, n)
            intercambio_LIC = True

        #recibimos la solicitud de descifrado por parte del CDM
        signature = recv(clientSocketCDM, KEY_CDM, IV_CDM)
        print("***Mensaje firmado recibido")

        #reenviamos a serv_lic el mensaje creado por CDM, su kpub y la solicitud
        msj = signature+b"-----"+nombre_nuevo.encode()
        send(msj, clientSocketLIC, KEY_LIC, IV_LIC)
        print("---Solicitud de licencia firmada enviada")

        send(pemCDM, clientSocketLIC, KEY_LIC, IV_LIC)
        print("---Clave pública CDM enviada")
        resp = recv(clientSocketLIC, KEY_LIC, IV_LIC)

        if resp==b"ERROR":
            print("***ERROR en la obtencion de la licencia")
        else:
            print("***Licencia recibida")
            #enviamos las credenciales de acceso
            send(resp, clientSocketCDM, KEY_CDM, IV_CDM)
            print("---Licencia enviada a CDM")

#Comando QUIT----------------
elif comando_l[0] == 'QUIT':
    #Intercambio de clave simetrica----------------------
    if intercambio_CDM==False:
        clientSocketCDM.connect(TCPserverAddrCDM)
        rsa_cliente(pem, clientSocketCDM, d, n)
        intercambio_CDM = True

    if intercambio_LIC==False:
        clientSocketLIC.connect(TCPserverAddrLIC)
        rsa_cliente(pem, clientSocketLIC, d, n)
        intercambio_LIC = True

    send(b"quit", clientSocketCDM)
```

```
            send(b"quit", clientSocketLIC)
            break

        else:
            print("400 ERROR. MENSAJE NO IDENTIFICADO")

    except Exception as e:
        print('\nERROR')
        print(e)

        #borra contenido de temp
        if ("claves.txt" or "clave_de_claves.txt") in os.listdir("temp/"):
            os.remove("temp/claves.txt")
            os.remove("temp/clave_de_claves.txt")

print("Client has finished its work")
clientSocket.close()
```

3.4. CDM

```
################################
#                              #
#              CDM             #
#                              #
################################

#LIBRERIAS--------------------------------------------------------------------
import os
from socket import *
from cryptography.hazmat.primitives import serialization, hashes, padding
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.primitives.asymmetric import rsa, padding

#FUNCIONES--------------------------------------------------------------------
from funciones import rsa_server, sign_message, send, recv

#CONFIGURACION SOCKETS--------------------------------------------------------
serverPort = 60002
serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind(('', serverPort))
serverSocket.listen()

print('CDM ready to receive connections')
connectionSocket, clientAdress = serverSocket.accept()

conectado = False

#CONFIGURACION CIFRADO RSA ----------------------------------------------------
intercambio = False
KEY = os.urandom(16)
IV = os.urandom(16)
```

```
private_key = rsa.generate_private_key(
    public_exponent=65537,
    key_size=2048,
)
public_key = private_key.public_key()

pem = public_key.public_bytes(
   encoding=serialization.Encoding.PEM,
   format=serialization.PublicFormat.SubjectPublicKeyInfo
)

firmado=False

#cifrador CTR
aesCipher_CTR = Cipher(algorithms.AES(KEY),modes.CTR(IV))

#PROGRAMA-----------------------------------------------------------------------
while True:
    try:
        #Intercambio de clave simetrica----------------------
        if intercambio==False:
            rsa_server(KEY, IV, connectionSocket)
            intercambio = True

        #Comunicacion con cliente----------------------------
        #recibimos mensaje del cliente con el nombre del fichero
        msj = recv(connectionSocket, KEY, IV)
        print(msj)

        #si el mensaje es quit --> para el programa
        if msj == b"quit":
            break

        nombre_nuevo = msj.decode() #decodifica el mensaje
        print("\n***Mensaje recibido:",nombre_nuevo)

        send(pem, connectionSocket, KEY, IV) #envia al cliente su clave publica

        #guarda datos de fichero cifrado
        f = open('ficheros_recibidos/'+nombre_nuevo, 'rb')
        data = f.read()
        f.close()

        #firmamos la solicitud y se la enviamos al cliente
        try:
            message_to_sign = msj
            signature = sign_message(private_key, message_to_sign)
            send(signature, connectionSocket, KEY, IV)
            print("---Mensaje firmado enviado")

        except Exception as e:
            print("\nERROR. Mensaje firmado NO enviado")
            print(e)
            break
```

```
        #recibimos las credenciales de acceso al contenido multimedia
        resp = recv(connectionSocket, KEY, IV)

        if resp!=b"ERROR": #comprobamos que no haya dado error
            print("***Licencia recibida")

            #sacamos KEY e IV de la respuesta del servidor
            KEY2 = resp[:16]
            IV2 = resp[16:]

            print("***Clave del fichero recibida:",KEY2)
            print("***IV del fichero recibido:",IV2)

            #creamos descifrador para desencriptar el fichero
            aesCipher_CTR2 = Cipher(algorithms.AES(KEY2),modes.CTR(IV2))
            aesDecryptor_CTR2 = aesCipher_CTR2.decryptor()

            #desencriptamos el fichero y lo guardamos
            data_decrypt = aesDecryptor_CTR2.update(data) + aesDecryptor_CTR2.finalize()

            nombre = nombre_nuevo.split('.')
            nom = nombre[0][:-2]+"."+nombre[1]
            f_decrypt = open('ficheros_recibidos/'+nom, "wb")
            f_decrypt.write(data_decrypt)
            f_decrypt.close()

            os.remove('ficheros_recibidos/'+nombre_nuevo) #borramos el fichero encriptado
            print("***Fichero desencriptado")
        else:
            print("\nERROR al desencriptar")
            break

    except Exception as e: #en caso de error: lo imprimimos y notificamos al cliente
        print('\nERROR')
        print(e)
        aesEncryptor_CTR = aesCipher_CTR.encryptor()
        msg_enc = aesEncryptor_CTR.update(b"ERROR") + aesEncryptor_CTR.finalize()
        send(msg_enc, connectionSocket, KEY, IV)

print("\nServer has finished its work")
connectionSocket.close()
serverSocket.close()
```

3.5. Funciones:

Este es un programa python con funciones que hemos utilizado en el resto de programas. Menos las funciones de conversión entre enteros y bits, las hemos desarrollado nosotros para simplificar el código de los demás programas.

```
import os
```

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

#Cifrador CTR para ficheros-----------------------------------------------------
def cifra_CTR(nombre):
    #clave y vector de inicializacion
    KEY = os.urandom(16)
    IV = os.urandom(16)

    #para que no haya retornos de carro en KEY o IV
    while (b"\n" in KEY):
        KEY = os.urandom(16)
    while (b"\n" in IV):
        IV = os.urandom(16)

    #cifrador CTR
    aesCipher_CTR = Cipher(algorithms.AES(KEY),modes.CTR(IV))
    aesEncryptor_CTR = aesCipher_CTR.encryptor()

    #saber si vamos a cifrar un fichero o el txt de claves
    if nombre[-3:]!='txt':
        #datos
        nombre_l = nombre.split('.')
        img = open('ficheros_server/temp/'+nombre, 'rb')
        data = img.read()
        img.close()

        #encriptamos en modo CTR
        data_encrypt = aesEncryptor_CTR.update(data)

        #guardamos el fichero encriptado
        fhand = open('ficheros_server/temp/'+nombre_l[0]+'_e.'+nombre_l[1], 'wb')
        fhand.write(data_encrypt)
        fhand.close()

        #escribimos KEY e IV en el fichero de claves
        fhand = open('temp/claves.txt', 'ab+')
        fhand.write(nombre.encode('utf-8')+b'---'+KEY+b'---'+IV)
        fhand.close()
    else:
        #leemos el txt
        f = open('temp/claves.txt','rb')
        data = f.read()
        f.close()

        #encriptamos el contenido y borramos el fichero original
        aesEncryptor_CTR = aesCipher_CTR.encryptor()
        data_encrypt = aesEncryptor_CTR.update(data)+ aesEncryptor_CTR.finalize()
        os.remove("temp/claves.txt")

        #guardamos el fichero encriptado
        fhand = open('temp/claves.txt', 'wb')
```

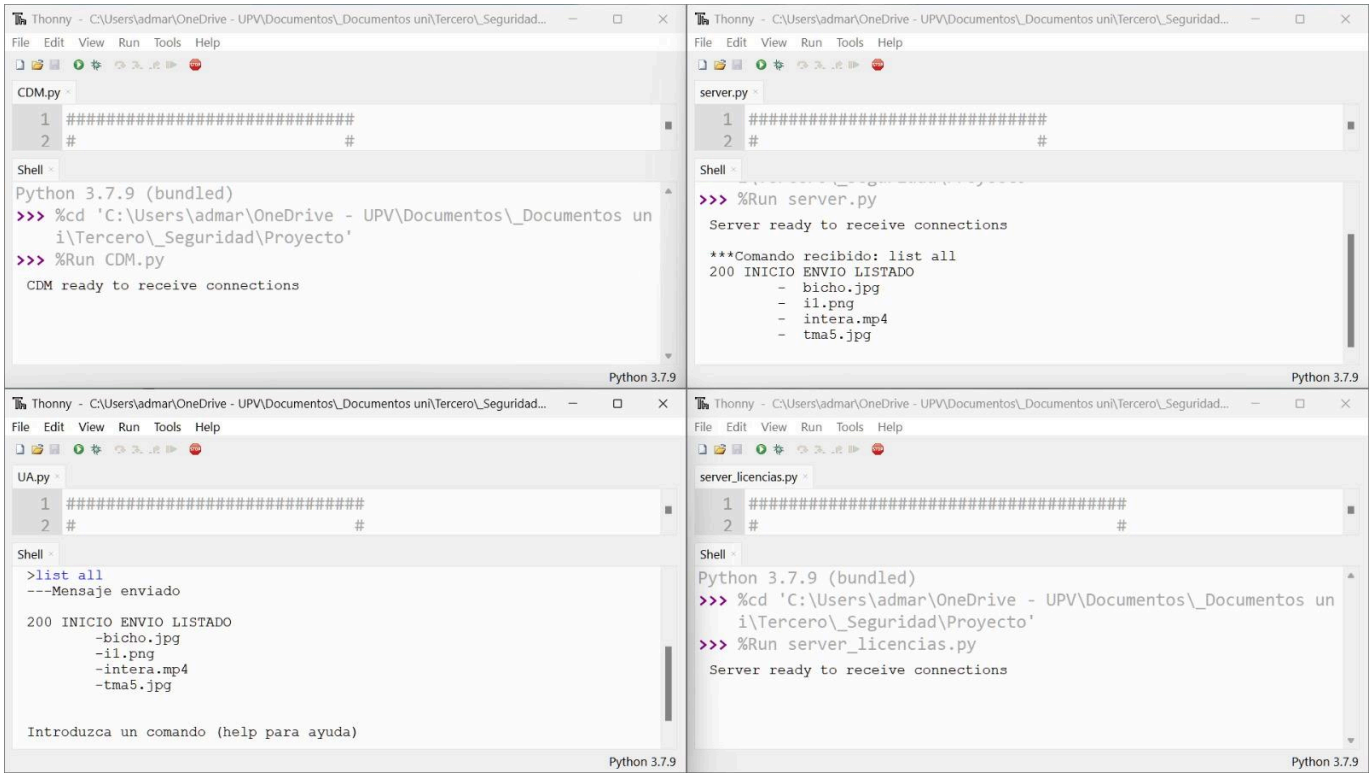
```
        fhand.write(data_encrypt)
        fhand.close()

        #guardamos la clave en otro txt
        fhand = open('temp/clave_de_claves.txt','wb')
        fhand.write(KEY+b'\n')
        fhand.write(IV)
        fhand.close()

#Conversion bytes <--> int-----------------------------------------------------
def bytes_to_int(b):
    return int.from_bytes(b, byteorder='big')

def int_to_bytes(i):
    return i.to_bytes((i.bit_length()+7)//8, byteorder='big')

#Funciones para cifrado asimetrico RSA-----------------------------------------
def rsa_cliente(pem, clientSocket, d, n):
    #envia la clave publica
    clientSocket.send(pem)

    #recibe KEY
    key_cifrada = clientSocket.recv(2048)

    #recibe IV
    iv_cifrado = clientSocket.recv(2048)

    #descifra KEY con clave privada
    key_dec_int = pow(bytes_to_int(key_cifrada), d, n)
    KEY = int_to_bytes(key_dec_int)

    #descifra IV con clave privada
    iv_dec_int = pow(bytes_to_int(iv_cifrado), d, n)
    IV = int_to_bytes(iv_dec_int)

    print("***Clave simétrica recibida")

    return KEY, IV

def rsa_server(KEY, IV, connectionSocket):
    #recibe clave publica
    pem = connectionSocket.recv(2048)
    kpub_cliente = serialization.load_pem_public_key(pem)
    print("***Clave pública recibida")

    #parametros de la clave publica
    e = kpub_cliente.public_numbers().e
    n = kpub_cliente.public_numbers().n

    #cifra KEY con la clave publica
    KEY_int = bytes_to_int(KEY)
    key_cifrada = int_to_bytes(pow(KEY_int, e, n))

    #cifra IV con la clave pública
```

```
    IV_int = bytes_to_int(IV)
    iv_cifrado = int_to_bytes(pow(IV_int, e, n))

    #envia KEY+IV cifrados
    connectionSocket.send(key_cifrada)
    connectionSocket.send(iv_cifrado)
    print('---Clave simétrica enviada')

#Firma digital-----------------------------------------------------------------
def sign_message(private_key, message):
    signature = private_key.sign(
        message,
        padding.PSS(
            mgf=padding.MGF1(hashes.SHA256()),
            salt_length=padding.PSS.MAX_LENGTH
        ),
        hashes.SHA256()
    )
    return signature

#Funcion para encriptar y enviar mensajes--------------------------------------
def send(msg, clientSocket, KEY="", IV=""):
    #comprueba si el mensaje a enviar se quiere encriptar
    if msg == b"quit":
        msg_enc = msg
    else:
        #crea cifrador
        aesCipher_CTR = Cipher(algorithms.AES(KEY),modes.CTR(IV))
        aesEncryptor_CTR = aesCipher_CTR.encryptor()

        #si el mensaje es string lo convierte a bytes
        if type(msg)==str:
            msg = msg.encode()

        #encripta el mensaje
        msg_enc = aesEncryptor_CTR.update(msg) + aesEncryptor_CTR.finalize()

    #envia el mensaje
    clientSocket.send(msg_enc)

#Funcion para recibir y desencriptar mensajes----------------------------------
def recv(connectionSocket, KEY="", IV=""):
    #recibe el mensaje
    msg_enc = connectionSocket.recv(2048)

    #comprueba si el mensaje es quit
    if msg_enc == b"quit":
        print(msg_enc)
        return msg_enc

    #crea el descifrador
    aesCipher_CTR = Cipher(algorithms.AES(KEY),modes.CTR(IV))
    aesDecryptor_CTR = aesCipher_CTR.encryptor()
```

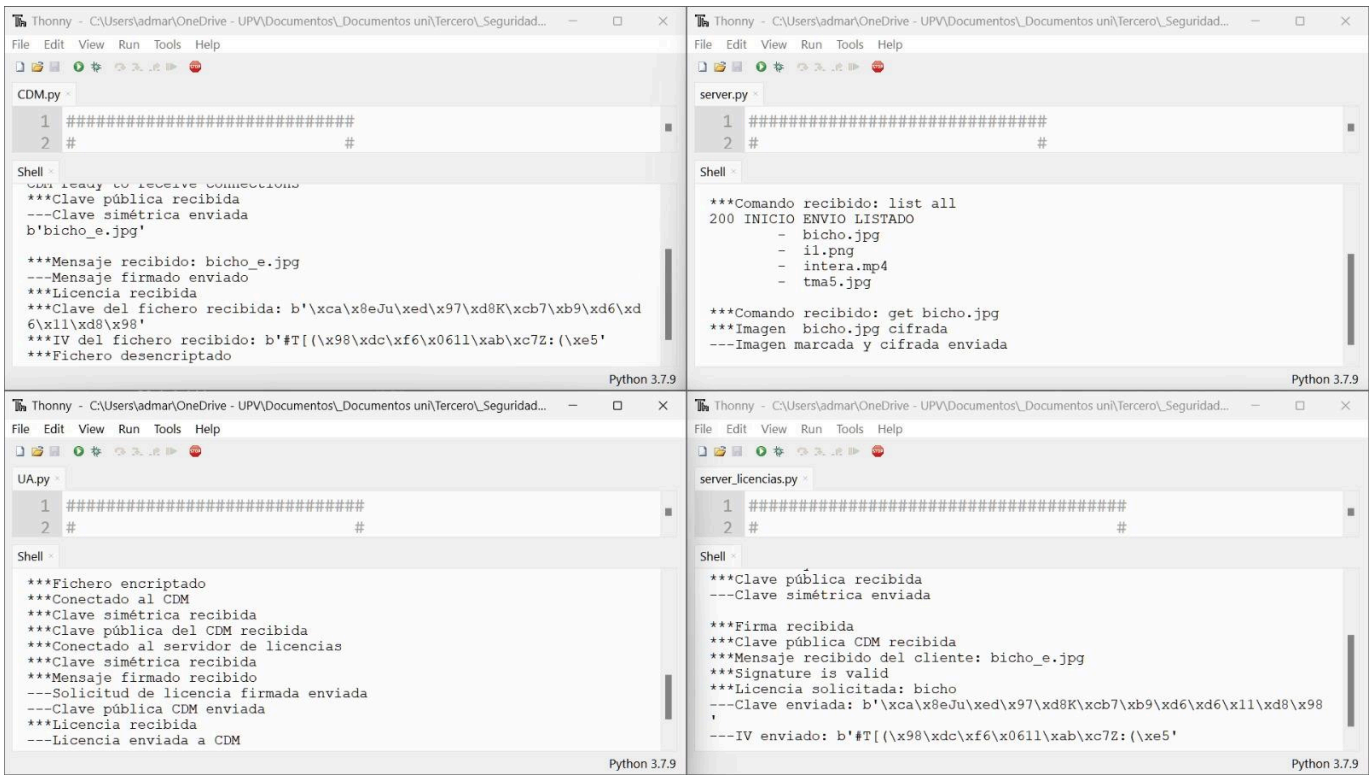
```
#desencripta el mensaje
msj = aesDecryptor_CTR.update(msg_enc) + aesDecryptor_CTR.finalize()
return msj
```

3.6.- Capturas funcionamiento

Comando list all:



Comando get <fichero>:



4.- ORGANIZACIÓN DEL EQUIPO

La organización que hemos tenido durante la realización del trabajo ha sido la siguiente:

Comenzamos realizando la comunicación básica de servidores y clientes entre todo el equipo mediante reuniones presenciales.

Una vez tuvimos una buena comunicación entre los servidores y el cliente principal dividimos el trabajo para que fuese más rápido durante las vacaciones de Navidad:

- Aiden se encargó del cifrado de la comunicación entre el cliente y el servidor de licencias.
- Adrián añadió las marcas de agua.
- Paula realizó el cifrado del txt de las claves.
- Ángela implementó la firma digital.

Además, durante las vacaciones hicimos reuniones online (en Teams) para poner en común el trabajo realizado y ayudar a aquellos que hubieran tenido algún problema con su parte asignada.

Seguidamente juntamos todas las partes realizadas individualmente, y decidimos repartir de nuevo lo que quedaba de trabajo:

- Aiden y Adrián se encargaron de separar el cliente en CDM y UA.
- Ángela se encargó del vídeo.
- Paula realizó la memoria.

5.- CONCLUSIONES

Al llegar al final de nuestro proyecto, no podemos evitar sentir un gran sentido de logro y aprendizaje. Esta experiencia nos brindó una comprensión más profunda de la asignatura, llevándonos a explorar la comunicación entre servidores y clientes, así como a sumergirnos en las diversas formas de cifrado y descifrado.

Lo más destacado de este viaje fue, sin duda, la maravillosa sinfonía de trabajo en equipo que logramos crear. Juntos, no solo enfrentamos desafíos técnicos, sino que también nos apoyamos mutuamente, aprendimos de nuestros errores y celebramos

cada pequeño triunfo. Esta colaboración no sólo impulsó nuestro conocimiento técnico, sino que también fortaleció nuestros lazos como equipo.
Estamos realmente orgullosos del camino recorrido y emocionados por las lecciones que llevamos hacia el futuro.