TAREA EVALUABLE

Criptografía aplicada y Esteganografía

1. Encuentre la mejor solución para dada una clave de usuario de 128

bits generar 10 claves que permitan cifrar 10 ficheros, cada uno con una clave diferente. Implemente la solución para cifrar y descifrar dichos ficheros (su contenido no es relevante). Pista: Openssl y scripts le ayudarán en esta tarea.

```
Para generar claves a partir de otra clave, primero se debe verificar la entropía de la misma, puesto que, en dependencia de si es alta o baja, debemos usar un algoritmo u otro. Teniendo en cuenta que es una clave de usuario, podemos asumir que es de baja entropía, por lo tanto, no es recomendable usar algoritmos como HKDF. Además, no es necesario utilizar esquemas jerárquicos ni se especifica que se quiera para almacenamiento o rotaciones, por lo tanto, no es necesario usar algoritmos como un KDF jerárquico. Por lo anterior mencionado, decido usar Argon2.
```

Implemento 2 soluciones usando python y OpenSSL con los resultados de salida en las carpertas solucion_1_python y solicion_1 respectivamente. Además previamente tengo los 10 documentos a cifrar/desifrar en una carpeta docs dentro de cada una de las carpetas mencionadas anteriormente.

Solución 1 con python

Para solucionar este ejercicio uso las bibliotecas de python cryptography y argon2 para los cifrados, adicionalmente uso pandas para mostrar los resultados en formato de tabla al final de la ejecución. Inicialmente creo variables para la calve de usuario que al no ser especificada creo una cualquiera de 16 bytes. el salt usado para los cifrados y la carpeta donde estann los dicumentos y donde voy a poner las salidas de los cifrados y descifrados. Luego hago una limpieza de las carpetas para asegurar que existen las carpetas y que no queden ficheros duplicados. Luego he creado una función para derivar las n claves usando el algoritmo argon2 usando la clave y salt creado; además he creado las funciones para cifrar y descrifrar que tienen por entrada la clave a usar, la ruta al archivo a ejecutar el cifrado/descifrado y la ruta de salida. Posteriormente derivo 10 claves usando el algoritmo argon2 y guardo estas 10 claves en variables. Luego Recorro la cantidad de ficheros a cifrar y ejecuto mi funciondem cifrado con una clave diferente para cada uno dejandop los resultados en docs_cifrados, luego hago el mismo procedimiento esta vez para descifrar y guardo los documentos en docs_descifrados los que sonexactamente iguales a los que se encuentran en la carpeta docs. Finalmente pongo lso datos de los resultados en un data frame de pandas y lo imprimo en consola.

```
import os
import base64
from pathlib import Path
```

```python
from argon2.low_level import hash_secret_raw, Type
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms,
modes
from cryptography.hazmat.backends import default_backend
import pandas as pd
import shutil

# Configuración
USER_KEY = b"\x01" * 16  # 128 bits
SALT = b"somesalt1234567890"  # 16 bytes
BASE_DIR = Path("solucion_1_python")

FILES_DIR = BASE_DIR / "docs"
ENCRYPTED_DIR = BASE_DIR / "docs_cifrados"
DECRYPTED_DIR = BASE_DIR / "docs_descifrados"
N_KEYS = 10

os.makedirs(ENCRYPTED_DIR, exist_ok=True)
os.makedirs(DECRYPTED_DIR, exist_ok=True)

# Limpiar los directorios de cifrados y descifrados
for d in (ENCRYPTED_DIR, DECRYPTED_DIR):
    if os.path.exists(d):
        shutil.rmtree(d)
    os.makedirs(d, exist_ok=True)

# Selecciona los primeros 10 ficheros
all_files = [f for f in os.listdir(FILES_DIR) if
os.path.isfile(os.path.join(FILES_DIR, f)) and not f.endswith('.lnk')]
selected_files = all_files[:N_KEYS]

# Deriva 10 claves únicas usando Argon2
def derive_keys(user_key, salt, n):
     keys = []
     for i in range(n):
          salt_i = salt + i.to_bytes(2, 'big')
          key = hash_secret_raw(
               secret=user_key,
               salt=salt_i,
               time_cost=2,
               memory_cost=65536,
               parallelism=2,
               hash_len=32,
               type=Type.I
          )
          keys.append(key)
     return keys

def encrypt_file(key, in_path, out_path):
     with open(in_path, 'rb') as f:
```

```
            data = f.read()
      iv = os.urandom(16)
      cipher = Cipher(algorithms.AES(key), modes.CFB(iv),
backend=default_backend())
      encryptor = cipher.encryptor()
      ct = encryptor.update(data) + encryptor.finalize()
      with open(out_path, 'wb') as f:
            f.write(iv + ct)

def decrypt_file(key, in_path, out_path):
      with open(in_path, 'rb') as f:
            raw = f.read()
      iv, ct = raw[:16], raw[16:]
      cipher = Cipher(algorithms.AES(key), modes.CFB(iv),
backend=default_backend())
      decryptor = cipher.decryptor()
      pt = decryptor.update(ct) + decryptor.finalize()
      with open(out_path, 'wb') as f:
            f.write(pt)

# Derivar claves
keys = derive_keys(USER_KEY, SALT, N_KEYS)

# Cifrar archivos a docs_cifrados
encrypted_files = []
for i, fname in enumerate(selected_files):
      in_path = os.path.join(FILES_DIR, fname)
      out_path = os.path.join(ENCRYPTED_DIR, f"{fname}.enc")
      encrypt_file(keys[i], in_path, out_path)
      encrypted_files.append(out_path)

# Descifrar archivos a docs_descifrados
decrypted_files = []
for i, enc_path in enumerate(encrypted_files):
      dec_path = os.path.join(DECRYPTED_DIR, selected_files[i])
      decrypt_file(keys[i], enc_path, dec_path)
      decrypted_files.append(dec_path)

# Imprimir tabla de claves y ficheros cifrados
df = pd.DataFrame({
      'Archivo original': selected_files,
      'Clave (hex)': [base64.b16encode(k).decode() for k in keys],
      'Archivo cifrado': [os.path.basename(f) for f in
encrypted_files],
      'Archivo descifrado': [os.path.basename(f) for f in
decrypted_files]
})
print(df.to_string(index=False))
```

```
                    Archivo original
Clave (hex)                          Archivo cifrado
Archivo descifrado
                             123.txt
59743960B7C3BEE0B170BB6E193ABE54FC8DBCC0D88537EEEFCBAE397C1A95BE
123.txt.enc                                  123.txt
        Documento sin título pdf.pdf
A347FC3EEF0F0D42BD4519A208FF4496E9E304F001F7D68590E0C69F6D3D7E33
Documento sin título pdf.pdf.enc         Documento sin título pdf.pdf
      Documento sin título word.docx
D7FC3D282A83B1AD7F1C5090928B7C0D93486A9954DB5BC71E1DCCCF2F642AA4
Documento sin título word.docx.enc       Documento sin título
word.docx
                         download.jpeg
AC9E1706086EA228D6BBCFA14109473A4F1052A2D0C58CE02E2932C75A523948
download.jpeg.enc                            download.jpeg
                          download.png
4150A24283F90F2BAB7554F517D6371316C99239B6331647CDE737170102107A
download.png.enc                             download.png
              New Archivo WinRAR.rar
D3C88EC33387E41ACBB5F8EA692190D6FA74DE64975D27EDB5CCC6C6A6F3BBC1
New Archivo WinRAR.rar.enc               New Archivo WinRAR.rar
                        New folder.zip
900980E9154145AB9BD24A4323D78C13C3C271B780DC6943CEBB53C9E2575F5B
New folder.zip.enc                           New folder.zip
  New Microsoft Excel Worksheet.xlsx
9DE5816959AF31C3A491A92468AA072B7CC448CB234431C953DB3327AB4280C9   New
Microsoft Excel Worksheet.xlsx.enc   New Microsoft Excel
Worksheet.xlsx
New Microsoft Publisher Document.pub
841A623485A82DC8CE49F34469A3E7526E11C64E6D7A8DAC077217DC2A28F70C New
Microsoft Publisher Document.pub.enc New Microsoft Publisher
Document.pub
  tumblr_m984tzdEG51rplctno1_500.gif
4774D2382173EDBDDF29F2A1BB480075F2CED25D15EDD8816B0CDAB9C2E26F5E
tumblr_m984tzdEG51rplctno1_500.gif.enc
tumblr_m984tzdEG51rplctno1_500.gif
```

Solución 1 con OpenSSL

Para solucion con OpenSSL he seguido las sugerencias en el enunciado del ejercicio y he creado dos scripts en bash que ejecutan comandos en consola usando OpenSSL.

- cifrar.sh: Este script al igual que hice en python inicialmente crea las variables de clave de usuario,carpetas de salida, etc. Luego se recorren todos los documentos en la carpeta para solucion_1 y en cada iteracion se genera un salt aleatorio cualquiera, se usa para generar una llave, luego se genera un IV usado para el cifrado del documento, adicionalmente en la carpeta donde se guarda el dicumento cifrado también se guarda un archivo meta con el salt e IV generado los que serviran para

generar de vuelta la clave de cifrado y descifrar el documento. Al recorrer 10 documentos se generan 10 claves de derivadas de la principal

- descifrar.sh: Este script de forma similar al visto previmente define las variables de clave de usuariuo. Luego recorre todos archicos meta donde se encuentra el salt e IV para generar de vuelta las claves de cifrado para cada dicumento, posteriormente usando el salt se genera la clave y con IV se hace el proceso de decifrado de los documentos guardando la salida en la carpeta de descifrados. Al final de la ejecución del script se tendrán los 10 documentos dscrifrados que seran exactamente igual a los originales.

Para ejecutarlos solo tiene que situarse en la carpeta ´Tarea 1´, abrir la consola y ejecutar los dos comandos que muestro abajo

```
./solucion_1/cifrar.sh
./solucion_1/descifrar.sh
```

Scripts

```
# cifrar.sh

#!/bin/bash
set -e

# ===== CONFIGURACIÓN =====
USER_KEY_HEX="00112233445566778899aabbccddeeff"  # Clave maestra de
128 bits
DOCS_DIR="solucion_1/docs"
OUTPUT_DIR="solucion_1/cifrados"
OPENSSL="openssl.exe"
export OPENSSL_CONF="C:\Program Files\OpenSSL-Win64\bin\openssl.cfg"

mkdir -p "$OUTPUT_DIR"

echo "Cifrando ficheros con Argon2id + AES-256-CBC"
echo "Clave maestra (hex): $USER_KEY_HEX"
echo "-----------------------------------------"

i=1
for file in "$DOCS_DIR"/*; do
  filename=$(basename "$file")
  echo "Archivo #$i: $filename"

  # Generar salt aleatorio (16 bytes)
  SALT=$($OPENSSL rand -hex 16)

  # Derivar clave con Argon2id (32 bytes)
  KEY_HEX=$($OPENSSL kdf -keylen 32 \
    -kdfopt pass:hex:$USER_KEY_HEX \
    -kdfopt salt:hex:$SALT \
```

```
    -kdfopt iter:3 \
    -kdfopt memcost:65536 \
    -kdfopt lanes:1 \
    ARGON2ID | xxd -p -c 256)

  # Generar IV aleatorio (16 bytes para AES-CBC)
  IV_HEX=$($OPENSSL rand -hex 16)

  # Cifrar con AES-256-CBC
  "$OPENSSL" enc -aes-256-cbc \
    -in "$file" \
    -out "$OUTPUT_DIR/$filename.enc" \
    -K "$KEY_HEX" \
    -iv "$IV_HEX" \
    -nosalt

  # Guardar metadatos (salt + iv)
  {
    echo "salt=$SALT"
    echo "iv=$IV_HEX"
  } > "$OUTPUT_DIR/$filename.meta"

  echo "Cifrado completado → $OUTPUT_DIR/$filename.enc"
  echo "Salt: $SALT"
  echo "Clave derivada: ${KEY_HEX:0:16}..."
  echo "-----------------------------------------"
  ((i++))
done

echo "Cifrado completado: todas las claves derivadas con Argon2id"

# descifrar.sh

#!/bin/bash
set -e

USER_KEY_HEX="00112233445566778899aabbccddeeff"
DOCS_DIR="solucion_1/cifrados"
OUTPUT_DIR="solucion_1/descifrados"
OPENSSL="openssl.exe"
export OPENSSL_CONF="C:\Program Files\OpenSSL-Win64\bin\openssl.cfg"

mkdir -p "$OUTPUT_DIR"

echo "Descifrando ficheros con Argon2id + AES-256-CBC"

for meta in "$DOCS_DIR"/*.meta; do
  base=$(basename "$meta" .meta)
  enc="$DOCS_DIR/$base.enc"

  # Leer metadatos
```

```
  source "$meta"

  # Derivar clave igual que al cifrar
  KEY_HEX=$($OPENSSL kdf -keylen 32 \
    -kdfopt pass:hex:$USER_KEY_HEX \
    -kdfopt salt:hex:$salt \
    -kdfopt iter:3 \
    -kdfopt memcost:65536 \
    -kdfopt lanes:1 \
    ARGON2ID | xxd -p -c 256)

  echo "Descifrando $base ..."
  "$OPENSSL" enc -d -aes-256-cbc \
    -in "$enc" \
    -out "$OUTPUT_DIR/$base" \
    -K "$KEY_HEX" \
    -iv "$iv" \
    -nosalt

  echo "$base descifrado → $OUTPUT_DIR/$base.dec"
  echo "-----------------------------------------------"
done

echo "Todos los archivos fueron descifrados correctamente"
```

2. Utilizando Openssl cifre un fichero usando el algoritmo autenticado AES-GCM

Implemento 2 soluciones usando python y OpenSSL con los resultados de salida en las carpertas solucion_2_python y solicion_2 respectivamente.

Solución 2 con python

Para este ejercicio uso la biblioteca cryptography. Inicialmente genero la clave, el IV aleatoriamente, y creo mi fichero con un contenido de ejmeplo cualquiera. Luego ccargo el fichero creado anteriormente y lo encripto con aes-gcm y mis llaves generadas anteriormente,guardo el encriptado en un archivo. Por ultimo vuelvo a cargar el fichero cifrado y lo verifico descifrandolo e impimiendo en consola el resultado del archivo descifrado.

```
from cryptography.hazmat.primitives.ciphers.aead import AESGCM
import os

# generar clave e IV
key = os.urandom(32)
iv = os.urandom(12)

# crear carpeta si no existe
os.makedirs("solucion_2_python", exist_ok=True)
# crear fichero de ejemplo si no existe y escribir texto
```

```python
example_text = b"Texto de ejemplo para cifrar\n"
with open("solucion_2_python/fichero_ejercicio_2.txt", "wb") as f:
    f.write(example_text)

# leer el fichero
with open("solucion_2_python/fichero_ejercicio_2.txt", "rb") as f:
    data = f.read()

# cifrar
aesgcm = AESGCM(key)
ciphertext = aesgcm.encrypt(iv, data, None)

# guardar resultados
with open("solucion_2_python/fichero_ejercicio_2.enc", "wb") as f:
    f.write(ciphertext)

# verificar descifrado
plaintext = aesgcm.decrypt(iv, ciphertext, None)
print("Texto descifrado:", plaintext.decode())
```

```
Texto descifrado: Texto de ejemplo para cifrar
```

Solución 2 con OpenSSL

A continuacion los comandos para el cifrado y vertificació de un fichero usando el algoritmo aes-gcm con OpenSSL. Inicialmente creo un fichero con un texto cualquiera, luego usando OpenSSL, genero las claves para cifrar y las guardo en la carpeta. Solo queda cifrar el fichero y descrifrarlo para verificarlo

```bash
# Crear el fichero original
echo "Este es un mensaje secreto autenticado" >
solucion_2/fichero_ejercicio_2.txt

# Generar clave (32 bytes = 256 bits) e IV (12 bytes = 96 bits)
openssl rand -hex 32 > solucion_2/key.hex
openssl rand -hex 12 > solucion_2/iv.hex

# Cifrar con AES-256-GCM (modo autenticado)
openssl enc -aes-256-gcm \
  -in solucion_2/fichero_ejercicio_2.txt \
  -out solucion_2/fichero_ejercicio_2.enc \
  -K $(cat solucion_2/key.hex) \
  -iv $(cat solucion_2/iv.hex) \
  -nosalt \
  -tag solucion_2/tag.bin \
  -p

# Descifrar verificando autenticidad
openssl enc -d -aes-256-gcm \
```

```
  -in solucion_2/fichero_ejercicio_2.enc \
  -out solucion_2/fichero_ejercicio_2_dec.txt \
  -K $(cat solucion_2/key.hex) \
  -iv $(cat solucion_2/iv.hex) \
  -nosalt \
  -tag $(cat solucion_2/tag.bin)
```

> He presentado un error con openssl, al parecer las versiones que h instalado no soportan el algoritmo gcm y me devuelve el siguiente error: `enc: Multiple cipher or unknown options: -aes-256-gcm`. Esto me pasa a pesar de tener la version estable 3.5.4.

```
$ openssl -version
OpenSSL 3.5.4 30 Sep 2025 (Library: OpenSSL 3.5.4 30 Sep 2025)
```

> Por lo que aunque conozco que con los comandos anteriormente mostradps se puede solucionar el ejercicio 2 pues no lo puedo ejecutar correctamente en mi computador. Por lo tanto implemento una solución alternativa usando AES-CTR + HMAC-SHA256

En esta solución, creo inicialmente mi fichero y genero las claves de cifrado y autenticación. Luego usando OpenSSL cifro con AES-CTR, con elo obtengo un fichero cifrado, luego lo que hago es calcular HMAC del fichero encriptado para la autenticación. Finalmente, se hacen los procesos inversos, se verifica la HMAC del fichero encriptado para verificar la autenticación recalculando el HMAC, y se descifra el fichero con AES-CTR.

```
# Alternativa sin GCM


# Crear fichero original
echo "Este es un mensaje secreto autenticado" >
solucion_2/fichero_ejercicio_2.txt

# Generar claves e IV
openssl rand -hex 32 > solucion_2/key_enc.hex
openssl rand -hex 32 > solucion_2/key_mac.hex
openssl rand -hex 16 > solucion_2/iv.hex

# Cifrar con AES-256-CTR
openssl enc -aes-256-ctr \
  -in solucion_2/fichero_ejercicio_2.txt \
  -out solucion_2/fichero_ejercicio_2.enc \
  -K $(cat solucion_2/key_enc.hex) \
  -iv $(cat solucion_2/iv.hex) \
  -nosalt

# Calcular HMAC-SHA256 (autenticación)
( echo -n $(cat solucion_2/iv.hex) | xxd -r -p; cat
solucion_2/fichero_ejercicio_2.enc ) \
```

```
  | openssl dgst -sha256 -mac HMAC -macopt hexkey:$(cat
solucion_2/key_mac.hex) \
  > solucion_2/fichero_ejercicio_2.hmac

# Verificar HMAC (comparar con fichero)
( echo -n $(cat solucion_2/iv.hex) | xxd -r -p; cat
solucion_2/fichero_ejercicio_2.enc ) \
  | openssl dgst -sha256 -mac HMAC -macopt hexkey:$(cat
solucion_2/key_mac.hex)

# Descifrar si la autenticación es válida
openssl enc -d -aes-256-ctr \
  -in solucion_2/fichero_ejercicio_2.enc \
  -out solucion_2/fichero_ejercicio_2_dec.txt \
  -K $(cat solucion_2/key_enc.hex) \
  -iv $(cat solucion_2/iv.hex) \
  -nosalt

# Resultado:
# - fichero_ejercicio_2.enc  → cifrado
# - fichero_ejercicio_2.hmac → autenticación
# - fichero_ejercicio_2_dec.txt → descifrado
# - key_enc.hex, key_mac.hex, iv.hex → claves y vector usados
```

3. Utilizando Openssl cree un par de clave pública/privada con curva elíptica, elija cualquier curva, y firme un documento verificando posteriormente su firma.

`He implementado las soluciones usando la curva elíptica prime256v1.`

Implemento 2 soluciones usando python y OpenSSL con los resultados de salida en las carpertas solucion_3_python y solicion_3 respectivamente.

Solución 3 con python

Para esta solución uso la biblioteca cryptography. Incialmente creo la llave privada usando la curva elíptica SECP256R1 y la guardo en un fichero, luego usando esa llave privada se deriva una llave pública y se guarda en otro fichero. A continuación creo un archivo con un texto cualquiera, el que usaré para firmar y verificar la firma. Finalmente firmo el fichero creado con ECDSA y mi llave privada y lo guardo en un fichero destinado a contener la firma. Finalmente verifico la firma haciendo el proceso inverso usando mi llave pública y comparándolo con mi mensaje original.

```
from cryptography.hazmat.primitives.asymmetric import ec
from cryptography.hazmat.primitives import hashes, serialization
from cryptography.hazmat.primitives.asymmetric.utils import
decode_dss_signature, encode_dss_signature
from cryptography.exceptions import InvalidSignature
from cryptography.hazmat.backends import default_backend
import os
```

```python
# Asegurarse de que exista la carpeta
os.makedirs("solucion_3_python", exist_ok=True)

# Generar clave privada (curva prime256v1)
private_key = ec.generate_private_key(ec.SECP256R1(),
default_backend())

# Guardar clave privada en formato PEM
with open("solucion_3_python/clave_privada.pem", "wb") as f:
    f.write(
        private_key.private_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PrivateFormat.TraditionalOpenSSL,
            encryption_algorithm=serialization.NoEncryption(),
        )
    )

# Derivar y guardar la clave pública
public_key = private_key.public_key()
with open("solucion_3_python/clave_publica.pem", "wb") as f:
    f.write(
        public_key.public_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PublicFormat.SubjectPublicKeyInfo,
        )
    )

# Crear archivo a firmar
mensaje = b"Mensaje importante que quiero firmar"
with open("solucion_3_python/fichero_ejercicio_3.txt", "wb") as f:
    f.write(mensaje)

# Firmar el archivo con SHA256 + ECDSA
firma = private_key.sign(mensaje, ec.ECDSA(hashes.SHA256()))
with open("solucion_3_python/firma.bin", "wb") as f:
    f.write(firma)

print("✅ Firma creada y guardada en solucion_3_python/firma.bin")

# Verificar la firma
try:
    public_key.verify(firma, mensaje, ec.ECDSA(hashes.SHA256()))
    print("Verificación exitosa: la firma es válida.")
except InvalidSignature:
    print("Verificación fallida: la firma NO es válida.")
```

```
✅ Firma creada y guardada en solucion_3_python/firma.bin
Verificación exitosa: la firma es válida.
```

Solución 3 con OpenSSL

Usando OpenSSL en el primer comando creo la llave privada con la curva elíptica prime256v1 y la guardo en un fichero. El segundo comando usa mi llave privada para generar la llave pública y la guarda en otro fichero. Luego Creo mi archivo a firmar con un texto cualquiera. Posteriormente se firma el fichero con la llave privada y se guarda en un fichero. Por último se verifica el fichero firmado con la llave pública.

```
# 1. Generar clave privada
openssl ecparam -name prime256v1 -genkey -noout -out
solucion_3/clave_privada.pem

# 2. Derivar clave pública
openssl ec -in solucion_3/clave_privada.pem -pubout -out
solucion_3/clave_publica.pem

# 3. Crear un archivo a firmar
echo "Mensaje importante que quiero firmar" >
solucion_3/fichero_ejercicio_3.txt

# 4. Firmar el archivo
openssl dgst -sha256 -sign solucion_3/clave_privada.pem -out
solucion_3/firma.bin solucion_3/fichero_ejercicio_3.txt

# 5. Verificar la firma
openssl dgst -sha256 -verify solucion_3/clave_publica.pem -signature
solucion_3/firma.bin solucion_3/fichero_ejercicio_3.txt
```