UNIVERSIDAD RAFAEL LANDIVAR

FACULTAD DE INGENIERÍA

LICENCIATURA EN INGENIERIA EN INFORMÁTICA Y SISTEMAS

EXAMEN FINAL

ESTUDIANTE

1503724-BRAYAN BENJAMÍN RUIZ QUINILLA

DOCENTE

TRUJILLO MAZARIEGOS ANGEL RICARDO

CAMPUS DE QUETZALTENANGO SAN ALBERTO HURTADO S.J

QUETZALTENANGO, 14 DE NOVIEMBRE DE 2025

INGRESAR MENSAJE:

```
Seleccione una opcion: 1
Ingrese el mensaje: Hoy es un dia nublado
```

CALCULAR HASH FNV-1

```
Seleccione una opcion: 2
Hash FNV-1: 2312953269
```

COMPRIMIR MESAJE

```
Seleccione una opcion: 3
Seleccione el mwtodo de compresión (1: RLE, 2: Huffman): 2
Mensaje comprimido (Huffman): 10110011101011000000011110100010110111100011110110010100101110010111011111011
```

FIRMAR HASH CON CLAVE PRIVADA RSA

```
Seleccione una opcion: 4
Generando par de claves RSA...
Firma digital: 95e54a53566193a9a38c666c71af07ef70624ab58cc7a3b0f0a892e82fc8fda08ed1a551f0ec6088345b5abe9c5c2dd5e69423795ab2333ba6a6bb1dcbeac51bf8b2abeabc
975c13b91e7412a9ed9b5484309cbb7eb4115b80d6555e5f384bc8d1eda3f813f1be3d9e0032ce0e47cdfd57845634c5996228de71150436cab01f3f56819a2a2c96dec0a520496c4aedde843
f3a1dcea7ef526c7bf0312faf880be8ca908140332ea3b4813df3559d34a71b34f37c81e412fc95b9cb03943b3a78669792a50ff52332636319800cd7222df8fd534a847882e84903f06f2ba9
436744393c71a3e10bc8031d11cd9db6147d11596d74c8885a6f5fa11d658397832f
Clave pública: -----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA+4ewaj2fakP+UqBFmmtz
UqSLhJIpSUoA8m42B2tVv34ryYql6S47WC4X7Wn0SYMy0AJHtLOs3SBIu+/EyNYu
g6ejJ8/8AkP1hh8NYUtNqcaRXdY6kJIyRKA7ysBcgaHoLzcTdSDEFQkXMJgM5VNx
SmEiG+DG3kaLAxdWlvDTrnxZC4IE3ad+TrA6quga/bbZm9K78v7q94hsSdNuaYv8
sHPnb/0Kt9bJ0RsRZ+Y0iGxNQpl196KrKHGvCwyEfrbmLBKPW7Hd0VfV0b9lwGaX
XcPTT/JouEgq6yrMEuYJcO8KjfP69xi7wueQfPkko/vOmQX3UclW3Aw1/hW3gNuc
rwIDAQAB
-----END PUBLIC KEY-----
```

SIMULAR ENVÍO

```
Seleccione una opcion: 5
Enviando mensaje comprimido junto con la firma digital y clave publica...
```

DESCOMPRIMIR Y VERIFICAR FIRMA

```
Seleccione una opcion: 6
Mensaje descomprimido: 10110011101011000000011110100010110111100011110110010100101110010111011111011
Nuevo hash FNV-1: 1372252218
Mensaje alterado o firma no valida.
```

Link Repositorio de Git: https://github.com/Benjamin-Ruiz/EXAMEN-FINAL-ED-II

```
import hashlib
import heapq
import os
from collections import defaultdict
from Crypto.PublicKey import RSA
from Crypto.Signature import pkcs1_15
from Crypto.Hash import SHA256
from Crypto.Random import get_random_bytes

# Función de Hashing FNV-1
def fnv1_hash(message):
    hash = 0x811c9dc5
    for byte in message.encode('utf-8'):
        hash = (hash * 0x01000193) & 0xffffffff
        hash ^= byte
    return hash

# Comprimir el mensaje
def rle_compress(message):
    compressed = []
    prev_char = ''
    count = 1
    for char in message:
        if char == prev_char:
            count += 1
        else:
            if prev_char:
                compressed.append((prev_char, count))
            prev_char = char
```

```
            count = 1
    compressed.append((prev_char, count))
    return compressed

# Comprimir usando Huffman
def huffman_compress(message):
    freq = defaultdict(int)
    for char in message:
        freq[char] += 1

    heap = [[weight, [char, ""]] for char, weight in freq.items()]
    heapq.heapify(heap)

    while len(heap) > 1:
        lo = heapq.heappop(heap)
        hi = heapq.heappop(heap)
        for pair in lo[1:]:
            pair[1] = '0' + pair[1]
        for pair in hi[1:]:
            pair[1] = '1' + pair[1]
        heapq.heappush(heap, [lo[0] + hi[0]] + lo[1:] + hi[1:])

    huff_dict = {}
    for pair in heap[0][1:]:
        huff_dict[pair[0]] = pair[1]

    compressed_message = ''.join(huff_dict[char] for char in message)
    return compressed_message, huff_dict
```

```
# Firma con RSA
def sign_message(private_key, message_hash):
    signer = pkcs1_15.new(private_key)
    h = SHA256.new(message_hash.encode('utf-8'))
    signature = signer.sign(h)
    return signature

# Verificar firma digital
def verify_signature(public_key, message_hash, signature):
    verifier = pkcs1_15.new(public_key)
    h = SHA256.new(message_hash.encode('utf-8'))
    try:
        verifier.verify(h, signature)
        return True
    except (ValueError, TypeError):
        return False
def main():
    print(" M      E      N      U")
    print("1. Ingresar mensaje")
    print("2. Calcular hash FNV-1")
    print("3. Comprimir mensaje")
    print("4. Firmar el hash con RSA")
    print("5. Simular envIo")
    print("6. Descomprimir y verificar firma")
    print("7. Salir")

    message = ""
    hash_value = None
    compressed_message = None
```

```
    signature = None
    public_key = None
    private_key = None

    while True:
        option = input("\nSeleccione una opcion: ")

        if option == '1':
            message = input("Ingrese el mensaje: ")

        elif option == '2':
            hash_value = fnv1_hash(message)
            print(f"Hash FNV-1: {hash_value}")

        elif option == '3':
            compression_option = input("Seleccione el mwtodo de compresión (1: RLE, 2:
Huffman): ")
            if compression_option == '1':
                compressed_message = rle_compress(message)
                print(f"Mensaje comprimido (RLE): {compressed_message}")
            elif compression_option == '2':
                compressed_message, _ = huffman_compress(message)
                print(f"Mensaje comprimido (Huffman): {compressed_message}")

        elif option == '4':
            if not private_key:
                private_key = RSA.generate(2048)
                public_key = private_key.publickey()
                print("Generando par de claves RSA...")
```

```
            if hash_value:
                signature = sign_message(private_key, str(hash_value))
                print(f"Firma digital: {signature.hex()}")
                print(f"Clave pública: {public_key.export_key().decode()}")
            else:
                print("Por favor, calcule el hash FNV-1 primero.")

        elif option == '5':
            if compressed_message and signature and public_key:
                print("Enviando mensaje comprimido junto con la firma digital y clave publica...")
            else:
                print("Complete las etapas previas antes de simular el envío.")

        elif option == '6':
            if compressed_message and signature and public_key:
                decompressed_message = ''.join([char * count for char, count in
compressed_message]) if isinstance(compressed_message, list) else compressed_message
                print(f"Mensaje descomprimido: {decompressed_message}")

                new_hash_value = fnv1_hash(decompressed_message)
                print(f"Nuevo hash FNV-1: {new_hash_value}")

                if verify_signature(public_key, str(new_hash_value), signature):
                    print("Mensaje autentico")
                else:
                    print("Mensaje alterado")
            else:
                print("No se ha recibido un mensaje valido para verificar.")
```

```
        elif option == '7':
            print("Saliendo del programa.")
            break

if __name__ == '__main__':
    main()
```