Proyecto 1

Nombre de la materia: Intercomunicación y seguridad en redes

Alumno: Brayan Blancas Monsalvo

Matricula: 201965973

Docente: Josué Pérez Romero

Introducción

El propósito principal de este proyecto es comprender las bases de diferentes mecanismos de seguridad usados en las redes de comunicación y en otros ámbitos, como lo es el cifrado simetrico, firmas digitales e integridad y autenticación.

El desarrollo muestra fragmentos de código de cada una de las actividades solicitadas, capturas de pantalla para mostrar el resultado y de explicaciones acerca de ciertos procedimientos.

La conclusión concentra el aprendizaje obtenido al finalizar las tres actividades, expresando la importancia de estos conceptos en entornos reales.

Finalmente, se incluye un enlace al repositorio con el código fuente.

Desarrollo

1. Cifrado simétrico:

Implementan o utiliza una pequeña librería para simular el cifrado y descifrado de un mensaje corto usando un algoritmo simple por flujo (como el XOR con una clave) o por bloques (simulando AES con una librería si usas C++ o Python)

Demostración: Cifrar un mensaje con una clave y mostrar que solo el poseedor de la clave puede leerlo.

Código

Se definieron dos funciones: xor_encryp y xor_decrypt.

```
def xor_encrypt(message, key):
    message_encrypted = "".join(
        map(
            lambda ch: chr(ord(ch[1]) ^ ord( key[ch[0] % len(key)] ))
            ,
            enumerate(message)
        )
    )
```

```
    return message_encrypted

def xor_decrypt(ciphertext, key):
    return xor_encrypt(ciphertext, key)
```

xor_encrypt

La función xor_encryp recibe el mensaje y la clave para cifrar. Dentro del map se aplica la operación XOR (^) a cada letra del mensaje con una letra de la clave. La letra de la clave con la que se aplica el XOR se calcula mediante la operación módulo. Por ejemplo, el mensaje es 'hola' y mi clave es 'si', y se quiere aplicar XOR a la letra 'a' del mensaje con su correspondiente letra de la clave, se aplica un módulo entre el índice de la letra del mensaje % la longitud de la clave, quedando algo asi:

Letra mensaje	**Módulo**	**Letra clave**
h [0]	0 % 2 = 0	s [0]
o [1]	1 % 2 = 1	i [1]
l [2]	2 % 2 = 0	s [0]
a [3]	3 % 2 = 1	i [1]

De esta manera, se garantiza que cada letra de la clave se asigne de manera 'circular' a cada letra del mensaje.

Cabe aclarar que no se trata directamente con las letras de las clave y el mensaje, sino con su valor unicode (retorna un entero) con la función ord(), de esta manera se puede usar la operación XOR, ya que Python permite operaciones entre bits directamente con enteros.

Finalmente con los enteros Unicode encriptados, se convierten nuevamente en una cadena con su respectivo caractes que representa dicho número.

xor_decrypt

Para calcular el mensaje original, se aplica nuevamente la función XOR al mensaje encriptado con la misma clave.

Demostración

```python
def xor_decrypt(ciphertext, key):
    return xor_encrypt(ciphertext, key)


message = "MENSAJE ULTRASECRETO NO LEER"
key = "GANTZ"

cipher = xor_encrypt(message, key)
plain = xor_decrypt(cipher, key)

print("Mensaje original:", message)
print("Clave:", key)
print("Cifrado:", cipher.encode())
print("Descifrado:", plain)
```

[31] ✓ 0.0s Python

```
Mensaje original: MENSAJE ULTRASECRETO NO LEER
Clave: GANTZ
Cifrado: b'\n\x04\x00\x07\x1b\r\x04n\x01\x16\x13\x13\x0f\x07\x1f\x04\x13\x0b\x00\x15g\x0f\x01t\x16\x02\x04\x1c'
Descifrado: MENSAJE ULTRASECRETO NO LEER
```

El mensaje imprime incluso los caracteres que no se pueden imprimir (con .encode()). Para mejorar la presentación se puede mostrar en hexadecimal.

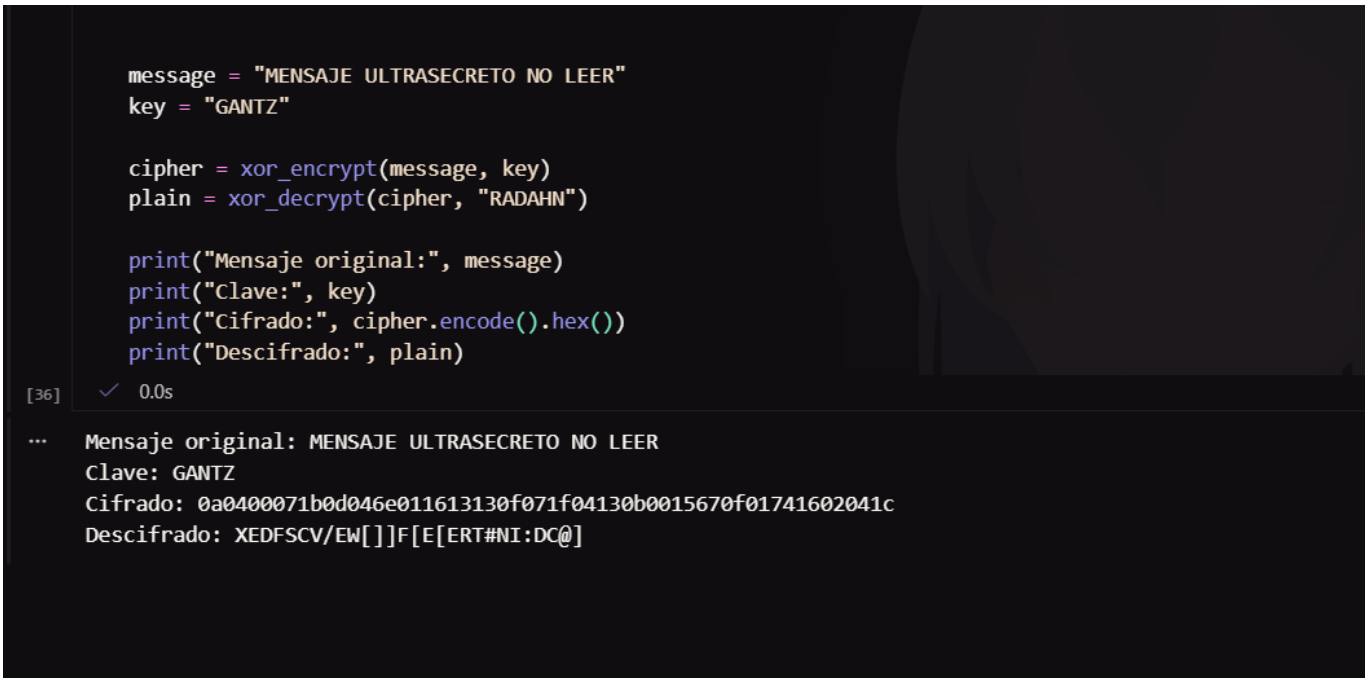
```python
message = "MENSAJE ULTRASECRETO NO LEER"
key = "GANTZ"

cipher = xor_encrypt(message, key)
plain = xor_decrypt(cipher, key)

print("Mensaje original:", message)
print("Clave:", key)
print("Cifrado:", cipher.encode().hex())
print("Descifrado:", plain)
```

[32] ✓ 0.0s Python

```
Mensaje original: MENSAJE ULTRASECRETO NO LEER
Clave: GANTZ
Cifrado: 0a0400071b0d046e011613130f071f04130b0015670f01741602041c
Descifrado: MENSAJE ULTRASECRETO NO LEER
```



Si se trata de desencriptar con una clave incorrecta, el mensaje original no se puede obtener.

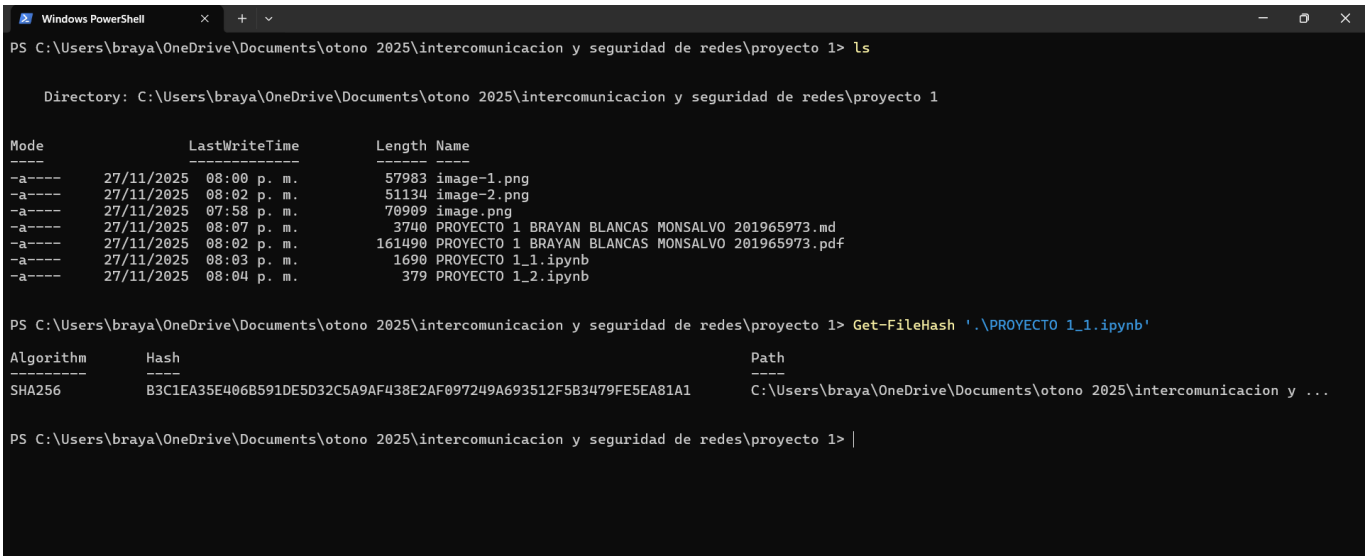
2. Funciones Hash y MAC (Integridad y Autenticación):

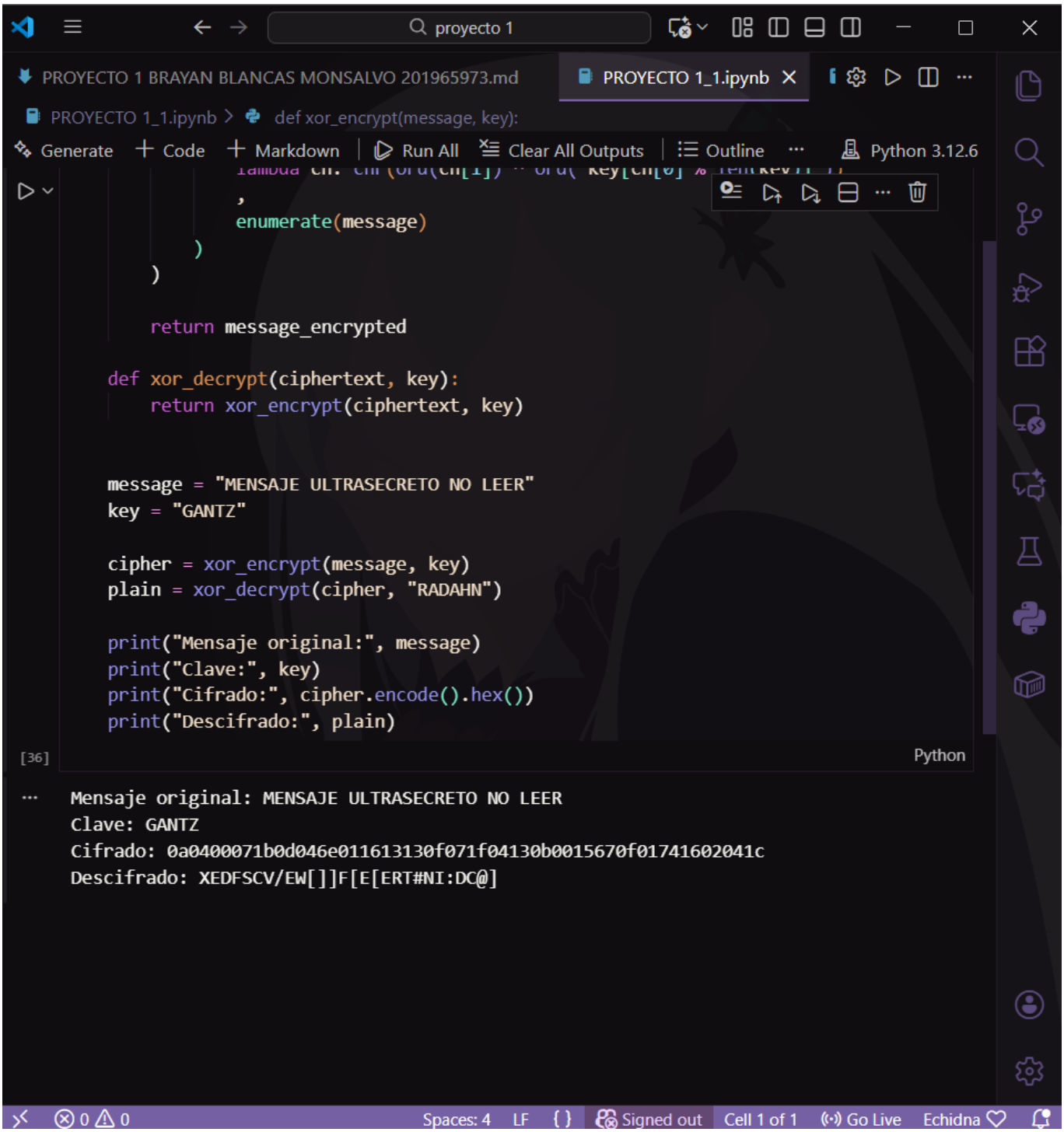
Utiliza una función Hash (ej. SHA-256) para calcular la huella digital de un archivo de texto.

Simula un ataque de integridad: Modifica el archivo de texto en un solo byte y recalcula el hash para demostrar como cambia drásticamente.

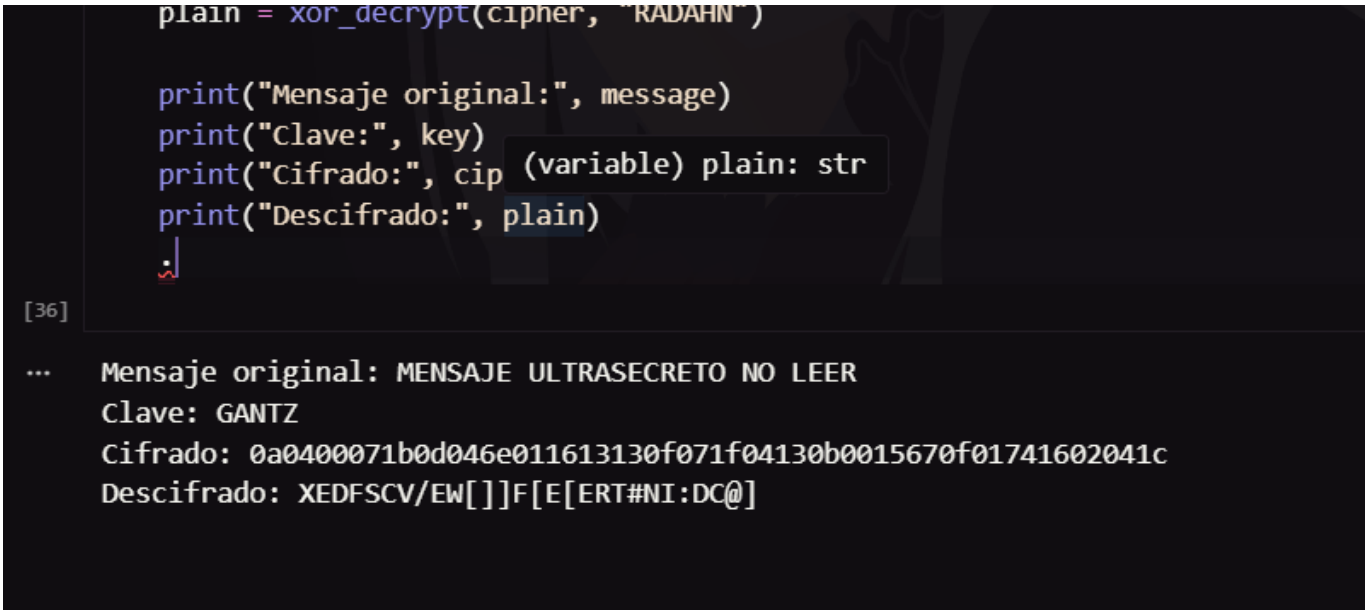
Función HASH

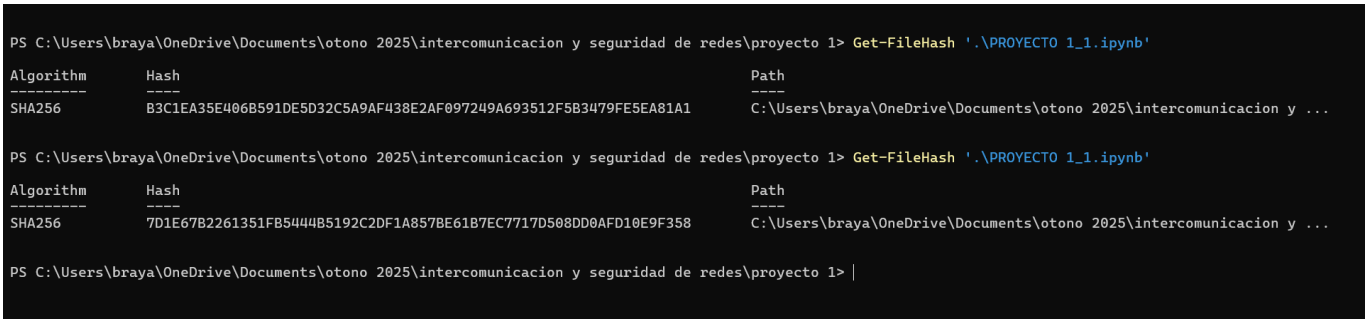
Para mostrar la huella digital de un archivo, se puede usar la función Get-FileHash de PowerShell que por defecto calcula el hash con el algoritmo SHA256.


Windows PowerShell
PS C:\Users\braya\OneDrive\Documents\otono 2025\intercomunicacion y seguridad de redes\proyecto 1> ls
Directory: C:\Users\braya\OneDrive\Documents\otono 2025\intercomunicacion y seguridad de redes\proyecto 1
Mode LastWriteTime Length Name
-a---- 27/11/2025 08:00 p. m. 57983 image-1.png
-a---- 27/11/2025 08:02 p. m. 51134 image-2.png
-a---- 27/11/2025 07:58 p. m. 70909 image.png
-a---- 27/11/2025 08:07 p. m. 3740 PROYECTO 1 BRAYAN BLANCAS MONSALVO 201965973.md
-a---- 27/11/2025 08:02 p. m. 161490 PROYECTO 1 BRAYAN BLANCAS MONSALVO 201965973.pdf
-a---- 27/11/2025 08:03 p. m. 1690 PROYECTO 1_1.ipynb
-a---- 27/11/2025 08:04 p. m. 379 PROYECTO 1_2.ipynb
PS C:\Users\braya\OneDrive\Documents\otono 2025\intercomunicacion y seguridad de redes\proyecto 1> Get-FileHash '.\PROYECTO 1_1.ipynb'
Algorithm Hash Path
SHA256 B3C1EA35E406B591DE5D32C5A9AF438E2AF097249A693512F5B3479FE5EA81A1 C:\Users\braya\OneDrive\Documents\otono 2025\intercomunicacion y ...
PS C:\Users\braya\OneDrive\Documents\otono 2025\intercomunicacion y seguridad de redes\proyecto 1>


proyecto 1
PROYECTO 1 BRAYAN BLANCAS MONSALVO 201965973.md
PROYECTO 1_1.ipynb
PROYECTO 1_1.ipynb > def xor_encrypt(message, key):
Generate
Code
Markdown
Run All
Clear All Outputs
Outline
Python 3.12.6
enumerate(message)
return message_encrypted
def xor_decrypt(ciphertext, key):
return xor_encrypt(ciphertext, key)
message = "MENSAJE ULTRASECRETO NO LEER"
key = "GANTZ"
cipher = xor_encrypt(message, key)
plain = xor_decrypt(cipher, "RADAHN")
print("Mensaje original:", message)
print("Clave:", key)
print("Cifrado:", cipher.encode().hex())
print("Descifrado:", plain)
[36]
Python
Mensaje original: MENSAJE ULTRASECRETO NO LEER
Clave: GANTZ
Cifrado: 0a0400071b0d046e01161313 0f071f04130b0015670f01741602041c
Descifrado: XEDFSCV/EW[]]F[E[ERT#NI:DC@]
Spaces: 4
LF
Signed out
Cell 1 of 1
Go Live
Echidna

Al realizar cualquier modificación, el hash cambia drásticamente.





Implementa un MAC (Message Authentication Code): Usa una clave compartida con la función hash (HMAC) para demostrar que se necesita tanto la clave como el mensaje para verificar la integridad y la autenticidad del remitente.

Código

El código es muy simple, se requiere calcular el hmac (con el módulo hmac y hashlib de Python) pasando los bytes del mensaje y la clave. Con el hash calculado se 'envia' y el receptor lo recibe. Después trata de verificar que el mensaje es autentico y no esta alterado. Para ello vuelvo a aplicar el hmac con el mensaje recibido + la clave que él y el emisor conocen. Si ambas HMAC coinciden el mensaje no fue alterado.

```
import hmac
import hashlib

mensaje = b"Mensaje importante"
clave_compartida = b"clave_secreta"

hmac_generado = hmac.new(clave_compartida, mensaje, hashlib.sha256).hexdigest()
print("HMAC generado:", hmac_generado)

hmac_verificacion = hmac.new(clave_compartida, mensaje,
hashlib.sha256).hexdigest()

if hmac_generado == hmac_verificacion:
```

```
    print("MENSAJE VERIFICADO")
else:
    print("MENSAJE ALTERADO")
```

Si el hash no coincide.

3. Firma digital (No repudio):

Investiga una libreria para generar una pareja de claves asimétricas (RSA o ECDSA).

Simulación de firma: Firma el hash del mensaje usando la clave privada y permite que cualquiera verifique la firma con la clave pública. Muestra como esto garantia el No Repudio.

Openssl es una herramienta de código abierto, entre muchas de sus caracteristicas, permite cifrar, descrifrar, firmar y verificar archivos y mensajes con RSA.

1. Crear una clave privada

RSA requiere de dos claves, una pública y otra privada. Con openssl se puede crear una clave pública a traves de una clave privada

```
openssl genrsa -out private_key.pem 2048
```

```
braya@asustuf MINGW64 ~/OneDrive/Documents/otono 2025/intercomunicacion y seguridad de redes/proyecto 1 (master)
$ openssl genrsa -out private_key.pem 2048

braya@asustuf MINGW64 ~/OneDrive/Documents/otono 2025/intercomunicacion y seguridad de redes/proyecto 1 (master)
$ cat private_key.pem
-----BEGIN PRIVATE KEY-----
MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQCceH9rYrwXYMjp
SgyGQM8EDuxNO+sGOTYJkSu1biX6B/Jji1l8f59znx4+HgGTZM/RJUOKOG44I2Ok
jzkVrDl53yj7TkauROJqeK7L8nt/u9vMA9sO7hT4Bs+SEuqnDDa6opJZgS9E9H3l
+9jG+T4OqJPUjBRakeMV8qLS72bmFOajHRqtOC7hCkFWFF3nk7iGTpGcLT8N3S7M
WS/zZtlVw2wckObyzOSjj3LCcKyjR5ZgkWwwdIp+4NvBoFICiqqRpzHTiYXaHiTM
ZyF/KEkyUjJSb3+9xnf6rzmfs2j9X+IBLwmqB+na+1y5s9h5ubQ1NWL3dedtcZs9
kMnkxSNdAgMBAAECggEAEK/cHOpOdOTL4G9UexvdyAfEZT9wvOLsk6QfBnQXCj1/
OOUDV97vdR/21HaUQfEaJaql9v2IB6YljWTV2GOb2xaOOupD6PGyEFNLBvaW+6Yf
+He/OLsEUvg8+k3YpxSLIKfWVkPL8/1kENslQQETQHJzg5Ml14vf7TcJbNf5wBc3
KQo9cIWHQ6TlDxjSTBbEk1wLmek4TOBLIq6Y8GxUogebSyqBXOja8IY+BTKgqCjB
oG/p2TxwniCoIOItHASUtB7hL1TTj6AJg86IO9zU4+zfHvx49dy9d4x+YvZU9ONC
BgI9n4yHkiOfIwD7mwoSBOiT5si1pUHfqs4ZCWZAyQKBgQDcClpXT7VOI7eQDyib
qwvH6q6sOOBDL6PCp6QlS+b1AUGcUbRd12J98NrH+aZK4hw4xo9FbqrLnPTO5XB4
wJPHbl3Z8teH8LxSI4aucvfPXrXgWL5fYqmwPWP8UgL3GUc/wz5ZyDn75iBJhewG
NZJg2UVLVnpQobJLkiOhkV/O9QKBgQC2CqF4JmDGNOvor5RqLXmC/rKyVvRfryLQ
RnN5QDVOCOYWIDOpTgbZG+l94+FmBTVfcbdro5ZOsthvgGVGZFLHKkwyvRoXdhMY
gVXawOUOtb2Fga2J8PVD5YyV6qYuv76E6Wcp4NQDtK1QyYA/MjVxb5EABKkymiJp
ABKJi3OzyQKBgFI6gDY+sFWdf1yTfZ5m1jk4Z1wuyQfyhqUYlsvlSILonEc5dUjY
U5My7d9rWQyE4POKnsHYsHTOy7uBnH1ToiYWBXaMPKUtfoLHie4pQpg5EYisQUBO
E5Gr73B8iWUkCi823Fca7yhGgGxZYrGTVYdhynZz87OsKX/6NdyMAir9AoGBAK+b
CbBRtUEAzqUj2tOiEnmfINqQryQuWjL4AySQIABTJUFqOBUsQ8bj+hWK4j/7/kSX
+gxHvhMkyOykWw3QcVnXH7JCYQOIxdr5yMxBD9oNBOnmknVij/ZzVW648J5gMXUd
gnsPevYoMJiiCKjtOkgfC4Os8dryfQc1Hqy4doqhAoGAUgwc8W4+dhAXxqZxfd2b
2A43OrKVe/wfjgptUSw25ZT9KUdNrSjFUlkJddrHVigdk3f2IMgHb69l8FPciVYu
pDpHD9OnfEwrDqE1x2mYi6CfB1Fjg2gXzBS43CqUZW9nICZ4r3FYn/9Vyl9ThWqx
mEi47/iOd2ga1hEA51ob/Eo=
-----END PRIVATE KEY-----
```

2. Crear una clave pública a partir de una clave privada

```
openssl rsa -in  private_key.pem -pubout -out public_key.pem
```

```
braya@asustuf MINGW64 ~/OneDrive/Documents/otono 2025/intercomunicacion y seguridad de redes/proyecto 1 (master)
$ openssl rsa -in  private_key.pem -pubout -out public_key.pem
writing RSA key

braya@asustuf MINGW64 ~/OneDrive/Documents/otono 2025/intercomunicacion y seguridad de redes/proyecto 1 (master)
$ cat public_key.pem
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAnHh/a2K8F2DI6UoMhkDP
BA7sTdPrBjk2CZErtW4l+gfyY4tZfH+fc58ePh4Bk2TPOSVNCtBuOCNtJI85Faw5
ed8o+O5GrkdCaniuy/J7f7vbzAPbNO4U+AbPkhLqpww2uqKSWYEvRPR95fvYxvk+
DqiT1IwUWpHjFfKiOu9m5hTmoxOarTgu4QpBVhRd55O4hk6RnCO/Dd0uzFkv82bZ
VcNsHJNG8szko49ywnCso0eWYJFsMHSKfuDbwaBSAoqqkacx04mF2h4kzGchfyhJ
MlIyUm9/vcZ3+q85n7No/V/iAS8Jqgfp2vtcubPYebmONTVi93XnbXGbPZDJ5MUj
XQIDAQAB
-----END PUBLIC KEY-----

braya@asustuf MINGW64 ~/OneDrive/Documents/otono 2025/intercomunicacion y seguridad de redes/proyecto 1 (master)
$
```

3. Calcular y firmar el hash del mensaje

```
openssl dgst -sha256 -sign private_key.pem -out firma.bin mensaje.txt
```

```
braya@asustuf MINGW64 ~/OneDrive/Documents/otono 2025/intercomunicacion y seguridad de redes/proyecto 1 (master)
$ openssl dgst -sha256 -sign private_key.pem -out firma.bin mensaje.txt

braya@asustuf MINGW64 ~/OneDrive/Documents/otono 2025/intercomunicacion y seguridad de redes/proyecto 1 (master)
$ ls
'PROYECTO 1 BRAYAN BLANCAS MONSALVO 201965973.md'   'PROYECTO 1_2.ipynb'   image-10.png   image-13.png   image-4.png   image-7.png   image.png         public_key.pem
'PROYECTO 1 BRAYAN BLANCAS MONSALVO 201965973.pdf'   firma.bin             image-11.png   image-2.png    image-5.png   image-8.png   mensaje.txt
'PROYECTO 1_1.ipynb'                                 image-1.png           image-12.png   image-3.png    image-6.png   image-9.png   private_key.pem

braya@asustuf MINGW64 ~/OneDrive/Documents/otono 2025/intercomunicacion y seguridad de redes/proyecto 1 (master)
$
```

4. Verificar la firma

```
openssl dgst -sha256 -verify public_key.pem -signature firma.bin mensaje.txt
```

```
braya@asustuf MINGW64 ~/OneDrive/Documents/otono 2025/intercomunicacion y seguridad de redes/proyecto 1 (master)
$ openssl dgst -sha256 -verify public_key.pem -signature firma.bin mensaje.txt
Verified OK

braya@asustuf MINGW64 ~/OneDrive/Documents/otono 2025/intercomunicacion y seguridad de redes/proyecto 1 (master)
$
```

Conclusión

Con estas tres actividades se demostraron los pilares de la criptografía y como estas trabajan en conjunto para garantizar la seguridad en la comunicación. La primera actividad mostró el cifrado simetrico y como solo quieren poseen la clave puede leer el contenido original. La segunda actividad demostró cómo las funciones hash son extremandamente sensibles al más mínimo cambio, y esto se aprovecha para garantizar que el contenido de un mensaje/archivo es autentico y no sufrió alguna alteración. Finalmente con la tercera actividad se generó una clave privada y clave pública, además se mostró que aquellos mensajes firmados con la clave original pueden ser verificados con su clave pública generada.

Estos conceptos en conjunto son la base de las comunicaciones, transacciones bancarias, validación de software, documentos legales digitales y una larga lista. A pesar de que las actividades se realizaron de manera simplificada, muestra los principios que protegen millones de interacciones en la red.

Enlace de GitHub

https://github.com/brayan-monsalvo4/proyectos-inter/tree/master/proyecto%201