

# 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 simétrico, 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\_encrypt 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_encrypt` 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 así:

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 caracteres 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

```
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.

```
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
```

```
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] ✓ 0.0s

```
... Mensaje original: MENSAJE ULTRASECRETO NO LEER
Clave: GANTZ
Cifrado: 0a0400071b0d046e011613130f071f04130b0015670f01741602041c
Descifrado: XEDFSCV/EW[]]F[E[ERT#NI:DC@]
```

Si se trata de descryptar 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
-----
SHA256         B3C1EA35E406B591DE5D32C5A9AF438E2AF097249A693512F5B3479FE5EA81A1
Path
C:\Users\braya\OneDrive\Documents\otono 2025\intercomunicacion y ...

PS C:\Users\braya\OneDrive\Documents\otono 2025\intercomunicacion y seguridad de redes\proyecto 1> |
```

```
PROJECTO 1 BRAYAN BLANCAS MONSALVO 201965973.md
PROJECTO 1_1.ipynb X
def xor_encrypt(message, key):
    lambda chr: chr(ord(chr[1]) ^ ord(key[chr[0]])) % 256
    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: 0a0400071b0d046e011613130f071f04130b0015670f01741602041c
Descifrado: XEDFSCV/EW[]]F[E[ERT#NI:DC@]
```

Al realizar cualquier modificación, el hash cambia drásticamente.

```
plain = xor_decrypt(cipher, "RADAHN")

print("Mensaje original:", message)
print("Clave:", key)
print("Cifrado:", cip (variable) plain: str
print("Descifrado:", plain)
~|

[36]

... Mensaje original: MENSAJE ULTRASECRETO NO LEER
Clave: GANTZ
Cifrado: 0a0400071b0d046e011613130f071f04130b0015670f01741602041c
Descifrado: XEDFSCV/EW[]]F[E[ERT#NI:DC@]

PS C:\Users\braya\OneDrive\Documents\otono 2025\intercomunicacion y seguridad de redes\proyecto 1> Get-FileHash '.\PROYECTO 1_1.ipynb'
Algorithm      Hash
-----
SHA256         B3C1EA35E406B591DE5D32C5A9AF438E2AF097249A693512F5B3479FE5EA81A1
Path
-----
C:\Users\braya\OneDrive\Documents\otono 2025\intercomunicacion y ...

PS C:\Users\braya\OneDrive\Documents\otono 2025\intercomunicacion y seguridad de redes\proyecto 1> Get-FileHash '.\PROYECTO 1_1.ipynb'
Algorithm      Hash
-----
SHA256         7D1E67B2261351FB5444B5192C2DF1A857BE61B7EC7717D508DD0AFD10E9F358
Path
-----
C:\Users\braya\OneDrive\Documents\otono 2025\intercomunicacion y ...

PS C:\Users\braya\OneDrive\Documents\otono 2025\intercomunicacion y seguridad de redes\proyecto 1> |
```

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")

```

```

if hmac_generado == hmac_verificacion:
    print("MENSAJE VERIFICADO")
else:
    print("MENSAJE ALTERADO")

```

[2] ✓ 0.0s

```

... HMAC generado: 220f0c7697dc266be77e2f9797428adcab7604ad598dc2cbf070d7a018f619d9
Integridad y autenticidad VERIFICADAS

```

Si el hash no coincide.

```

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, b"mensaje importante.", hashlib.sha256).hexdigest()

if hmac_generado == hmac_verificacion:
    print("MENSAJE VERIFICADO")
else:
    print("MENSAJE ALTERADO")

```

[8] ✓ 0.0s

```

... HMAC generado: 220f0c7697dc266be77e2f9797428adcab7604ad598dc2cbf070d7a018f619d9
MENSAJE ALTERADO

```

### 3. Firma digital (No repudio):

Investiga una librería 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 garantiza el No Repudio.

Openssl es una herramienta de código abierto, entre muchas de sus características, permite cifrar, descifrar, 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 través 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
SgyGQM8EDuxN0+sGOTYJkSu1biX6B/Jji1l8f59znx4+HgGTZM/RJU0KOG44I20k
jzkVrD153yj7TkauR0JqeK7L8nt/u9vMA9s07ht4Bs+SEuqndDa6opJZgS9E9H3l
+9jG+T40qJPujBRakeMV8qLS72bmFOajHRQtOC7hCkFWFF3nk7iGTpGcLT8N3S7M
WS/zZt1Vw2wck0byz0Sjj3LCcKyjR5ZgkwwdIp+4NvBoFICiqqRpzHTiYXaHiTM
ZyF/KEkyUjJSb3+9xnf6rzmfs2j9X+IBLwmqB+na+1y5s9h5ubQ1NLW3dedtcZs9
kMnkxSNdAgMBAECggEAEK/cHOP0d0TL4G9UexvdyAfeZT9wvOLsk6QfBnQXCj1/
00UDV97vdR/21HauQFEaJaq19v2IB6Y1jwTV2G0b2xa00upD6PGyEFNLBvaw+6Yf
+He/OLsEUvg8+k3YpxSLIKfwVkpL8/1kENS1QQETQHJzg5M114vf7TcJbNf5wBc3
KQo9cIW HQ67LDxjSTBbEk1wLmek4TOBLIq6Y8GxUogebSyqBX0ja8IY+BTkgqCjB
oG/p2TxwniCoIOiTHASUTB7hL1TTj6A3g86I09zu4+zfHvx49dy9d4x+YvZU90NC
BgI9n4yHki0fiWd7mwoSB0iT5si1pUHFqs4ZCWZAYQKBgQDcc1pXT7V0I7eQDyib
qvwH6q6s00BDL6PCp6Q1S+b1AUGcUbrd12J98NrH+aZK4hw4xo9FbqrLnPT05XB4
wJPBHl3Z8teH8LxS14aucvFPXrXgwL5FymwPWP8Ug13GUc/wz5ZyDn75iBJhewG
NZj2g2UvLVnpQobJLkiohkV/09QKKBgQC2CqF4JmDGN0vor5RqLXmC/rKyVvRfryLQ
RnN5QDV0COYwID0pTgbZG+194+FmBTVfcbdro5Z0sthvgGVGZFLHKkwyvRoXdhMY
gVxawOU0tb2Fga2J8PVD5YyV6qYuv76E6wcp4NQDtK1QyYA/MjVxb5EABKkyMiJp
ABKji30zyQKBGI6gDY+sFwdf1yTfZ5m1jk4Z1wuyQfyhQUY1sv1SLonEc5dUjY
U5My7d9rWQYe4P0KnsHYSHT0y7uBnH1ToiYWBXAMPKutfoLHie4pQpg5EYisQUBO
E5Gr73B8iWukC1823Fca/yhGgGxZYrGTvYdhynZz870sKX/6NdyMAir9AoGBAK+b
CbBRtUEAzqUj2t0iEnmfINqQryQuwJL4AySQIABTJUFq0BUusQ8bj+hWk4j/7/kSX
+gxHvhMky0ykww3QcVnXH7JCYQ0IxdR5yMxBd9oNB0nmknVij/ZzVW648J5gMXUd
gnsPevYoMjiCKjtk0gfc40s8dryfQc1Hqy4doqhAoGAUgwc8W4+dhAXxqZxfD2b
2A430rkVe/wfjgptU5w25ZT9KudNrsjFu1kddrHVigdk3f2IMgHb69l8FPciVYU
pDpHD90nfEwrDqE1x2mYi6CfB1Fjg2gXzBS43CqUZW9nICZ4r3FYn/9Vyl9ThWqx
mEi47/i0d2ga1hEA51ob/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
BA7sTdPrBjk2CErtw4l+gfYy4tZfH+fc58ePh4Bk2TP0SVNctBuOCNTJI85Faw5
ed8o+05GrkdCanuy/J7F7vzbAPbNO4U+AbPkhLqpwW2uqKSWYEvRPR95fvYxvk+
DqiTlIwUwpHjFfki0u9m5hTmox0arTgu4QpBVhRd5504hk6RnCO/Dd0uzFkv82bZ
VcNsHJNG8szko49ywnCso0eWYJFsMHSKfuDbaBSAoqqkacx04mF2h4kzGchfyhJ
M1IyUm9/vcZ3+q85n7No/V/iAS8Jqgfp2vtcubPYebm0NTvi93XnbXgbPZDJ5MUj
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 simétrico y como solo quienes poseen la clave pueden leer el contenido original. La segunda actividad demostró cómo las funciones hash son extremadamente sensibles al más mínimo cambio, y esto se aprovecha para garantizar que el contenido de un mensaje/archivo es auténtico 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.