Datos del Estudiante		Nota
Nombres: Lenin Mateo López Pozo		
Materia: Seguridad Informática		
Carrea: TI	**Fecha: 10.06.2025**	
Tarea Nro.	03	

AES: Estándar de Encriptación Avanzado

El Estándar de Encriptación Avanzado (AES) es un algoritmo de cifrado simétrico utilizado a nivel mundial para proteger información sensible. Fue estandarizado por el NIST en 2001 y desde entonces se ha convertido en el estándar confiable para cifrar archivos, comunicaciones y tráfico en redes.

AES es un cifrador por bloques. Cifra bloques fijos de 128 bits utilizando una clave que puede tener 128, 192 o 256 bits de longitud. El número de rondas de transformación depende del tamaño de la clave:

Con claves de 128 bits se realizan 10 rondas.

Con claves de 192 bits se realizan 12 rondas.

Con claves de 256 bits se realizan 14 rondas.

Cada ronda incluye las siguientes operaciones:

1. SubBytes: donde cada byte se sustituye mediante una tabla S-box.
2. ShiftRows: donde cada fila de la matriz se desplaza de forma circular.
3. MixColumns: donde cada columna se mezcla usando operaciones algebraicas.
4. AddRoundKey: donde los datos se combinan con una clave de ronda mediante XOR.

En la ronda final se omite el paso MixColumns. La descifrado aplica las operaciones inversas en orden inverso.

AES se utiliza ampliamente en diversas áreas:

Protege redes Wi-Fi (WPA2).

Asegura las comunicaciones a través de HTTPS y aplicaciones de mensajería.

Cifra el almacenamiento local como discos y memorias USB.

Salvaguarda bases de datos y credenciales de acceso.

Es fundamental en protocolos VPN para conexiones remotas seguras.

AES es eficiente, resistente a ataques de fuerza bruta y funciona bien tanto en hardware como en software. Cuando se implementa correctamente, sigue siendo uno de los algoritmos de cifrado más sólidos y confiables en la actualidad.

```python
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from Crypto.Util.Padding import pad, unpad

key = get_random_bytes(16) # 16-byte key (128 bits)
cipher = AES.new(key, AES.MODE_CBC)
plaintext = b"Mensaje secreto AES"
ciphertext = cipher.encrypt(pad(plaintext, AES.block_size))

# Decrypt
decipher = AES.new(key, AES.MODE_CBC, iv=cipher.iv)
decrypted = unpad(decipher.decrypt(ciphertext), AES.block_size)

print("Ciphertext:", ciphertext)
print("Decrypted:", decrypted.decode())
```

```
(venv) mateoprivate@Mac López-Tr3 % python3 src/aes_example.py
Ciphertext: b"~\x81$\xc2o\xac2'j\xee\x8a\xdcD\x0e\x06\x85\x92\xfc\xb9y\x93M\xff\xd5\xe6%\x1b\xb6\xb55\x9d\xb6"
Decrypted: Mensaje secreto AES
(venv) mateoprivate@Mac López-Tr3 %
```

Base58

Base58 es un esquema de codificación diseñado para transformar datos binarios (como bytes o enteros grandes) en una representación textual segura y legible. Se utiliza principalmente en Bitcoin para generar direcciones y claves privadas en un formato más amigable para humanos y sistemas.

Su alfabeto contiene 58 caracteres:

123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz

Se excluyen caracteres confusos como 0 (cero), O (letra mayúscula o), I (i mayúscula) y l (ele minúscula), así como los signos no alfanuméricos que aparecen en Base64.

El principio matemático es simple:

Para codificar, se convierte el dato binario en un entero.

Luego se aplica la base 58, dividiendo repetidamente entre 58 y tomando los residuos como índices del alfabeto.

Por ejemplo, 123456789 se convierte en “BukQL”.

Si el dato original tenía bytes con valor 0 al inicio (0x00), estos se representan como caracteres ‘1’ en Base58 para conservar la longitud real.

Para decodificar, se toma cada carácter del texto, se busca su índice en el alfabeto y se multiplica por potencias crecientes de 58 (de derecha a izquierda), reconstruyendo el valor original como entero, y luego como secuencia de bytes.

Base58 se utiliza en Bitcoin para representar:

Direcciones públicas (como 1AKDDsfTh8uY4X…)

Claves privadas WIF (como L4mee2GrpBSc…)

Claves extendidas xpub/xprv para wallets HD

Gracias a su diseño sin caracteres ambiguos y su facilidad de uso, Base58 es ideal para copiar, pegar o transmitir datos criptográficos sin errores visuales ni de formato.

```python
import base58

data = b"mensaje codificado"
encoded = base58.b58encode(data)
decoded = base58.b58decode(encoded)

print("Encoded Base58:", encoded.decode())
print("Decoded:", decoded.decode())
```

```
(venv) mateoprivate@Mac López-Tr3 % python3 src/base58_example.py
Encoded Base58: 5YAuu4YZ4h29aUe45m6iS5bMG
Decoded: mensaje codificado
(venv) mateoprivate@Mac López-Tr3 %
```

Fernet

Fernet funciona con claves de 32 bytes codificadas en base64 y proporciona tokens cifrados que incluyen marca de tiempo, vector de inicialización (IV), el contenido cifrado y un código HMAC de verificación.

Fernet es un esquema de cifrado simétrico que permite cifrar y descifrar mensajes con una sola clave secreta. Es parte de la biblioteca cryptography en Python, y está diseñado para ofrecer confidencialidad, integridad y autenticación en un solo paso, de manera sencilla y segura.

Para cifrar un mensaje:

Se genera una clave con Fernet.generate_key().

Se construye un objeto Fernet con la clave.

Se cifra el mensaje en bytes con .encrypt().

Para descifrar:

Se utiliza la misma clave y .decrypt() para obtener el texto original.

Si la integridad o la autenticidad fallan, lanza una excepción InvalidToken.

Fernet se basa internamente en AES-CBC con una clave de 128 bits, padding PKCS7, y HMAC-SHA256.

También permite derivar claves a partir de contraseñas usando KDFs como PBKDF2, con una sal (salt) segura y muchas iteraciones.

Gracias a su API clara y sus garantías criptográficas, Fernet es una opción confiable para proteger datos en memoria, archivos pequeños, o comunicaciones cifradas entre sistemas que comparten una clave secreta.

```python
from cryptography.fernet import Fernet

# Generate and save a key
key = Fernet.generate_key()
cipher = Fernet(key)

# Encrypt a message
message = b"Mensaje secreto con Fernet"
token = cipher.encrypt(message)

# Decrypt the message
decrypted = cipher.decrypt(token)

print("Encrypted:", token)
print("Decrypted:", decrypted.decode())

# Este código realiza los pasos clave de Fernet:
# Generación de clave: Fernet.generate_key() crea una clave secreta única.
# Inicialización del cifrador: Fernet(key) instancia un objeto Fernet con esa clave.
# Cifrado autenticado: .encrypt() produce un token Fernet seguro con marca de tiempo, IV y HMAC.
# Descifrado validado: .decrypt() asegura integridad y autenticidad antes de devolver el mensaje original.
# Este token es seguro, no puede ser manipulado ni leído sin la clave original, y es seguro para transmitirse o almacenarse temporalmente.
```

```
(venv) mateoprivate@Mac López-Tr3 % python3 src/fernet_example.py
Encrypted: b'gAAAAABoSJWgmFjOXp5Z5OBF4HbxfO5IQgO1KqKJBxb80ApmJvQFOJ-oUDfA9-XAq4y7rW2Wi_f0cRMw1Jh6XT2Npu99PSSapopUj3IDFFpEXKtH10pJvBA='
Decrypted: Mensaje secreto con Fernet
(venv) mateoprivate@Mac López-Tr3 %
```

RSA (con cryptography.hazmat)

RSA permite cifrar un mensaje con una clave pública y descifrarlo solo con la clave privada correspondiente. Este ejemplo utiliza la implementación de bajo nivel ofrecida por la biblioteca cryptography.

El bloque de código:

```python
from cryptography.hazmat.primitives.asymmetric import rsa, padding
from cryptography.hazmat.primitives import hashes

# Generate RSA keys
private_key = rsa.generate_private_key(public_exponent=65537, key_size=2048)
public_key = private_key.public_key()

message = b"Mensaje con RSA"

# Encrypt
ciphertext = public_key.encrypt(
    message,
    padding.OAEP(mgf=padding.MGF1(algorithm=hashes.SHA256()),
                 algorithm=hashes.SHA256(), label=None)
)

# Decrypt
plaintext = private_key.decrypt(
    ciphertext,
    padding.OAEP(mgf=padding.MGF1(algorithm=hashes.SHA256()),
                 algorithm=hashes.SHA256(), label=None)
)

print("Ciphertext:", ciphertext)
print("Decrypted:", plaintext.decode())
```

```
(venv) mateoprivate@Mac López-Tr3 % python3 src/rsa_example.py
Ciphertext: b"'\xf9\x1c\x93\x14\xc3EEb\xb6\xe5R?\x1f\x9f\xb8\x93\x80Ja\xd6t\xc7U{~Z\xbe',\xa8\xbe\x13;\x01C\x0c\x96\xe6g\x93\x08\xf8\x17\x95
\xe51\xbe\xe6\xa0\xb2n\xf9\xe2\x0e\xbe~a\xd1\xdf2\x9f\x1d1A\x82y'\x94\x90s\x12\xb1\x8e\xe2{#\xf4\x07t\x92\x82\xf4\xe8\xe3\xe0\\\xb8\x1d\x91\
xbb~\xd9\x9cF}\x15\x8d\xc1\xb5\xacm\xc3\x1e!\xce%\xd2\xcb\xb37{}}\xcc@\x0f\xdf\xe1\r\x10^o\xb5\x95\xd7k\x0c\xe7jI\x8e`\xb8\xe4\xc5\x9c\x8a4O
X\xe3v2\x9e-\xafe\xb9\xc1\xffw\xc1L\xc0\xd4\xe0\x8a\xd9\x18\x8e\xf6#N\xe2\x82\x82\xa11Y\x8b\xcb\xc0\xb2\x13\xf4u_\xb3\xd7\x18\xb1]\x1a\xe4\x
f2\xe2\x9f\x87\xaf\xd4\xeez^\xcdq\x8d\xf6y9/\xca\x9e\x06\x86\xb6\x7f\x83\xd0\xd6\xd8\xcb`X\xee\x11c\x87\xd7ge\x0b\xd6ac\xa4\x02\x86=\x9e\xe1
\xcet\xbe~\x99\x88\xb0\x1c\xb7\x03\xbd\xcd\xea\x90?\x10)\xd6\x90\x0f\xe7\xfdu\x0b\x04"
Decrypted: Mensaje con RSA
(venv) mateoprivate@Mac López-Tr3 %
```

Explicación por partes

se importan los módulos necesarios para generar claves RSA, definir el relleno OAEP y aplicar funciones hash (SHA-256)

rsa.generate_private_key(...) crea una clave privada con un exponente público estándar (65537) y tamaño de clave de 2048 bits

public_key = private_key.public_key() obtiene la clave pública asociada a la clave privada

message = b"Mensaje con RSA" define el mensaje original, codificado en bytes

ciphertext = public_key.encrypt(...) cifra el mensaje usando la clave pública

se utiliza OAEP (Optimal Asymmetric Encryption Padding) como esquema de relleno

se especifica que tanto la máscara como el hash principal usan SHA-256

plaintext = private_key.decrypt(...) descifra el mensaje con la clave privada

usa el mismo esquema de relleno y funciones hash para garantizar compatibilidad y seguridad

por último, se imprime el mensaje cifrado en bytes y el mensaje descifrado en texto plano

Este enfoque garantiza confidencialidad

ya que nadie fuera del dueño de la clave privada puede recuperar el mensaje original

la seguridad depende del tamaño de la clave y del uso adecuado de un esquema de relleno robusto como OAEP con SHA-256.

SHA-256 (Secure Hash Algorithm 256)

SHA-256 es un algoritmo criptográfico de resumen (hash) que genera una huella digital de 256 bits a partir de cualquier entrada. A diferencia de la encriptación, no puede revertirse para obtener el mensaje original. Se utiliza para verificar integridad, firmar digitalmente o validar contraseñas sin almacenarlas directamente.

```python
import hashlib

data = "mensaje de prueba"
hashed = hashlib.sha256(data.encode()).hexdigest()

print("SHA-256 Hash:", hashed)
```

```
(venv) mateoprivate@Mac López-Tr3 % python3 src/sha256_example.py
SHA-256 Hash: 789f289612bb711e6fda22a76c5b912a7832752d528ba2957af4b271137afdd8
(venv) mateoprivate@Mac López-Tr3 %
```

Se importa el módulo estándar hashlib, que contiene implementaciones de funciones de resumen como SHA-256.

data es el mensaje en texto plano que se desea convertir a un hash.

.encode() transforma la cadena en bytes, como requiere la función SHA-256.

hashlib.sha256(...) calcula el resumen criptográfico sobre los bytes del mensaje.

.hexdigest() convierte el resultado binario a una cadena legible en formato hexadecimal.

El valor producido

es un identificador único de 64 caracteres (256 bits)

cambia completamente con cualquier modificación mínima en el mensaje original

y no puede revertirse para recuperar el contenido original

Este tipo de función

es útil para verificar la integridad de datos,

comparar contraseñas sin guardarlas directamente

o firmar digitalmente documentos de forma segura

Referencias

Alexander Katz, Aloysius Ng, Patrick Bourg, Arron Kau, Christopher Williams, Eli Ross, & Jimin Khim. (2025). RSA Encryption. Brilliant.Org. https://brilliant.org/wiki/rsa-encryption/

Cryptography.io. (n.d.). Fernet (symmetric encryption). Cryptography.Io. Retrieved June 10, 2025, from https://cryptography.io/en/latest/fernet/

GeeksForGeeks. (2025, January 6). RSA Algorithm in Cryptography. GeeksForGeeks. https://www.geeksforgeeks.org/rsa-algorithm-cryptography/

GeeksforGeeks. (2025). Advanced Encryption Standard (AES). GeeksforGeeks. https://www.geeksforgeeks.org/advanced-encryption-standard-aes/

Gorelo Team. (2024, October 31). What is SHA256 Encryption: How it Works and Applications. Gorelo. https://www.gorelo.io/blog/sha256-encryption/

Greg Walker. (2025, May 31). Base58: An easy-to-share set of characters. Learn Me a Bitcoin. https://learnmeabitcoin.com/technical/keys/base58/

James A. St. Pierre. (2023). Advanced Encryption Standard (AES). The Federal Information Processing Standards Publication Series of the National Institute of Standards and Technology. https://doi.org/https://doi.org/10.6028/NIST.FIPS.197-upd1

Rahul Awati, Corinne Bernstein, & Michael Cobb. (2024, February 20). Advanced Encryption Standard (AES). TechTarget. https://www.techtarget.com/searchsecurity/definition/Advanced-Encryption-Standard

Rahul Gosavi. (2024, October 7). Encryption and Decryption with Fernet in Python. Medium. https://medium.com/@rahulgosavi.94/fernet-encryption-and-decryption-2101f0e3097a

Rahul Gosavi. (2024, October 7). Encryption and Decryption with Fernet in Python. Medium. https://medium.com/@rahulgosavi.94/fernet-encryption-and-decryption-2101f0e3097a

Sven VC. (2020, December 15). Understanding Base58 Encoding. Medium. https://medium.com/concerning-pharo/understanding-base58-encoding-23e673e37ff6