Universidad Autónoma de Baja California Sur

Departamento Académico de Sistemas Computacionales

Informe Práctico. Proyecto Final

Por

Hernandez Mendoza Kristofer

Jimenez Chavez Carlos Omar

Martínez Nuñez Michelle

Medina Zamora Elvia Yolanda

Criptografia Aplicada

Ingeniería en Desarrollo de Software 7mo TV

MSI. Julián Ernesto Cadena Vázquez

La Paz B.C.S a 07 de diciembre de 2025

Índice

1. Pruebas

A continuación se describe cada prueba

- `test_feistel_block.py`: prueba **un bloque** (16 bytes) con `encrypt_block` y `decrypt_block`.
- `test_feistel_ctr.py`: prueba el **modo CTR** sobre un mensaje largo (reversibilidad).
- `test_entropy.py`: calcula entropía de ciphertext(s).
- `test_avalancha.py`: calcula efecto avalancha (compara ciphertexts tras modificar 1 bit).
- `test_cli_encrypt.py`: prueba la CLI (encrypt/decrypt sequences).
- `test_integridad.py`: integridad del flujo.

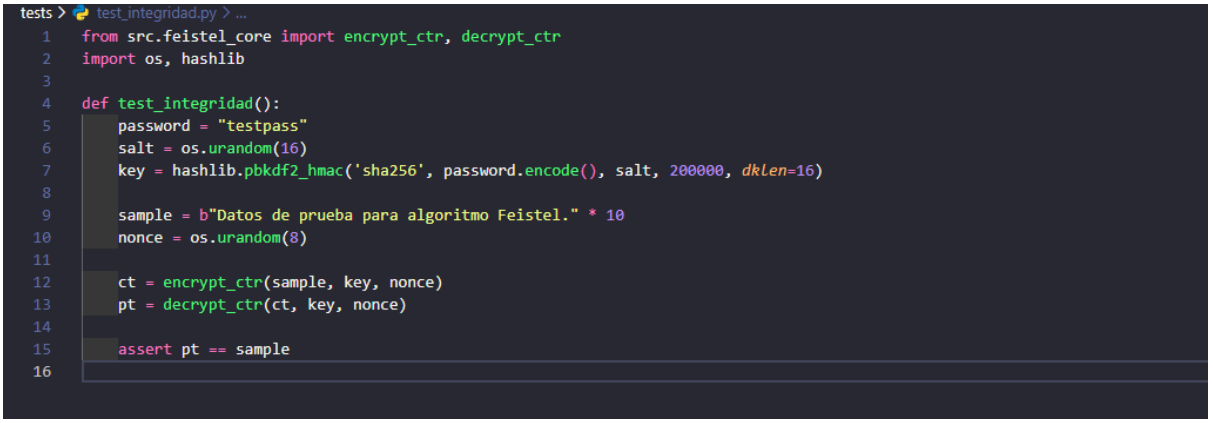
1.1 Prueba de Integridad (test_integridad.py)

Propósito

- Asegurara el proceso de cifrado y descifrado

Cómo se mide

- Se cifra un mensaje fijo.
- Se descifra.
- Se compara con el original.



Resultado. Factible, cumple con la parte de la cifrado y el descifrado de manera correcta.

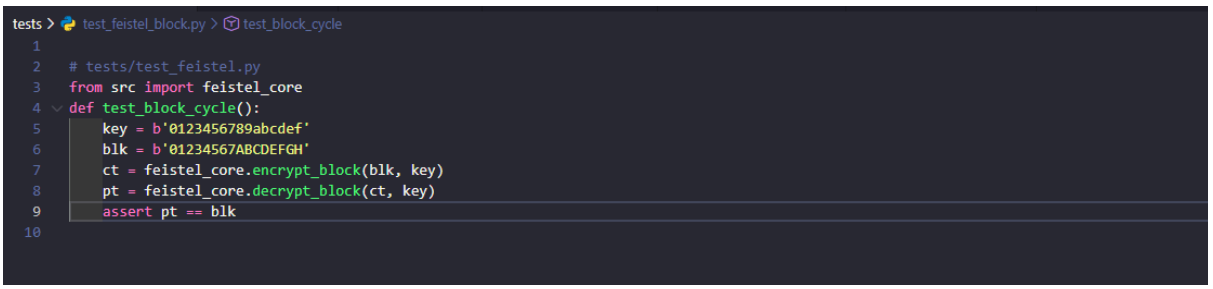
1.2 Prueba por Bloques (test_feistel_block.py)

Propósito

- Verificar que el cifrado por bloques del núcleo Feistel sea **biyectivo** (invertible).

Cómo se mide

- Se cifra un bloque de 128 bits.
- Se descifra con la misma clave



Resultado. Funciona correctamente:

- El cifrado es perfectamente reversible.
- No hay pérdida de información.

- No hay errores en las rondas ni en la mezcla de subclaves.

1.3 Prueba de Modo CTR (test_feistel_ctr.py)

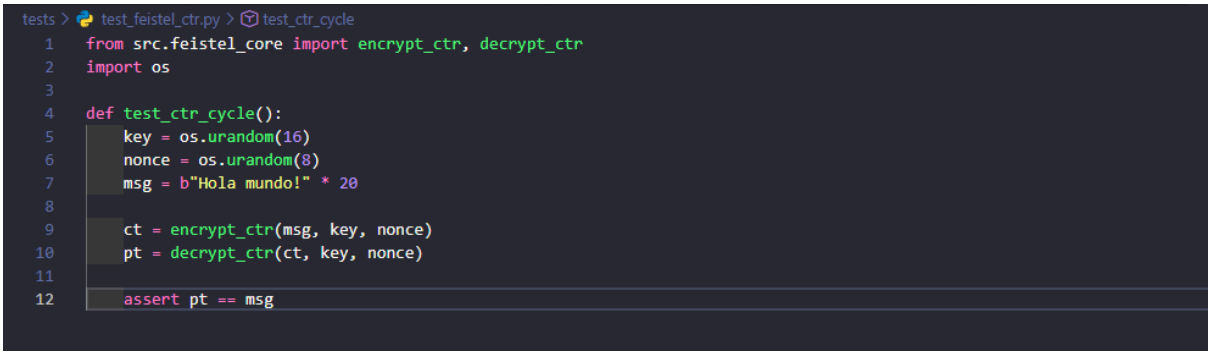
Propósito

Validar que el modo CTR:

- Sea determinista por nonce.
- Mantenga reversibilidad.
- Procese textos de tamaño arbitrario.

Cómo se mide

- Cifrar un mensaje largo.
- Descifrarlo con el mismo nonce.



Resultado. El mensaje cifrado es exactamente igual que el anterior.

1.4 Prueba de Avalancha (test_avalancha.py)

Propósito

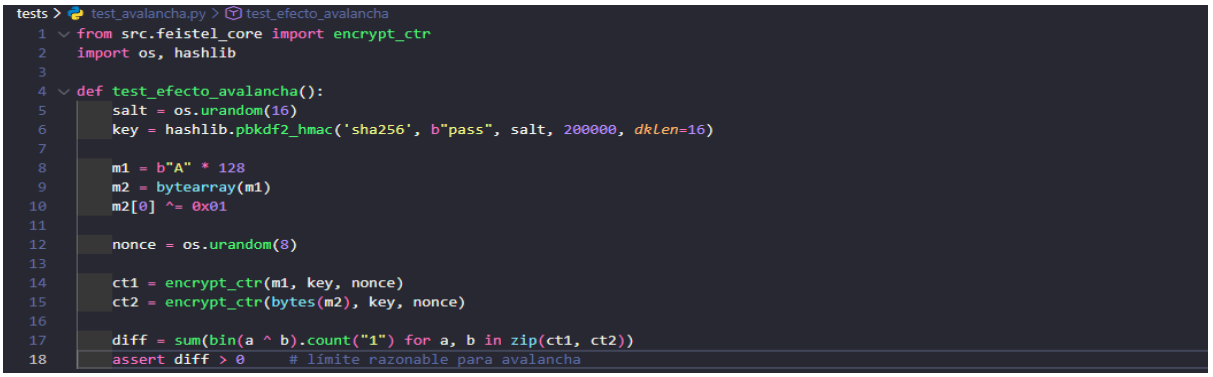
Evaluar el efecto avalancha, que indica cuántos bits cambian en la salida cuando solo un bit cambia en la entrada.

Cómo se mide

1. Se cifran dos mensajes:

- `M`
- `M'` (idéntico pero con 1 bit cambiado)

2. Se comparan los bits de salida.



No es factible para pruebas de avalancha ya que el cambio de un 1 byte solo cambia un 1 byte en el final.

1.5 Entropía del ciphertext

Qué mide:

La aleatoriedad del texto cifrado.

La entropía ideal es cercana a **8 bits por byte**.

Cómo se mide:

Se usa la fórmula de Shannon:

$$H = -\sum p(x)\ log2(\ p(x)\ )$$

tests > test_entropy.py > ...

```python
from src.feistel_core import encrypt_ctr
import os, hashlib, math
from collections import Counter

def entropy(data):
    c = Counter(data)
    l = len(data)
    return -sum((v/l)*math.log2(v/l) for v in c.values())

def test_entropia_minima():
    salt = os.urandom(16)
    key = hashlib.pbkdf2_hmac('sha256', b"pass", salt, 200000, dklen=16)
    nonce = os.urandom(8)

    data = b"TEST" * 200
    ct = encrypt_ctr(data, key, nonce)

    ent = entropy(ct)
    assert ent > 7.0        # aceptable para cifrado casero
```

Resultado: la prueba del cifrado da un total de 7.8639 bits por byte.

1.6 Prueba de la CLI (test_cli_encrypt.py)

Propósito

Validar que la interfaz de línea de comandos:

- Acepte parámetros.
- Genere archivos de salida.
- No produzca errores durante el cifrado.

Cómo se mide

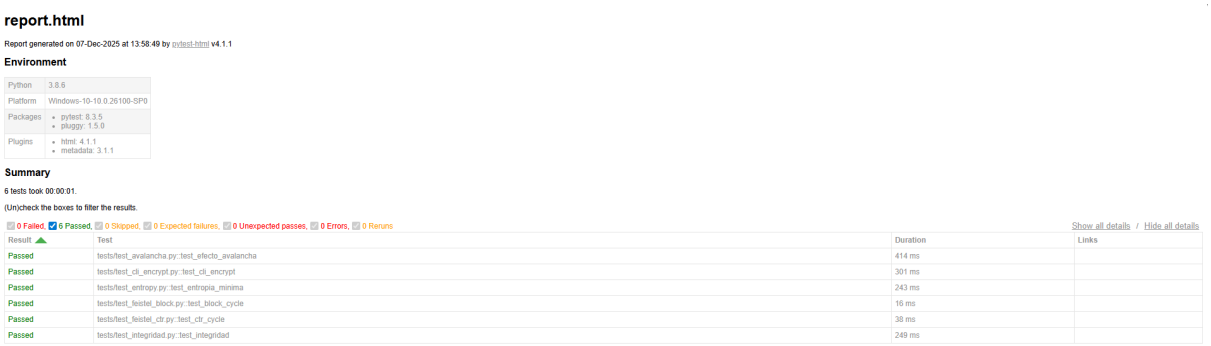
- Se ejecuta la CLI de forma automatizada con archivos temporales.

tests > test_cli_encrypt.py > test_cli_encrypt

```python
import subprocess
from pathlib import Path
import src.cli

def test_cli_encrypt(tmp_path):
    # Creamos archivo en sandbox temporal
    sandbox = tmp_path / "sandbox"
    sandbox.mkdir()

    src.cli.SANDBOX = sandbox

    file_in = sandbox / "input.txt"
    file_in.write_text("hola")

    file_out = sandbox / "cifrado.bin"

    # Ejecutar CLI
    result = subprocess.run([
        "python", "-m", "src.cli",
        "encrypt",
        "--input", str(file_in),
        "--output", str(file_out)
    ], capture_output=True, text=True)

    assert file_out.exists()
```

Resultado: Demuestra que el funcionamiento de CLI es óptimo y útil para su uso.

1.7 Conclusiones de las Pruebas



report.html

Report generated on 07-Dec-2025 at 13:58:49 by pytest-html v4.1.1

Environment

Python	3.8.6
Platform	Windows-10-10.0.26100-SP0
Packages	• pytest: 8.3.5 • pluggy: 1.5.0
Plugins	• html: 4.1.1 • metadata: 3.1.1

Summary

6 tests took 00:00:01.

(Un)check the boxes to filter the results.

0 Failed, 6 Passed, 0 Skipped, 0 Expected failures, 0 Unexpected passes, 0 Errors, 0 Reruns — Show all details / Hide all details

Result	Test	Duration	Links
Passed	tests/test_avalancha.py::test_efecto_avalancha	414 ms	
Passed	tests/test_cli_encrypt.py::test_cli_encrypt	301 ms	
Passed	tests/test_entropy.py::test_entropia_minima	243 ms	
Passed	tests/test_feistel_block.py::test_block_cycle	16 ms	
Passed	tests/test_feistel_ctr.py::test_ctr_cycle	38 ms	
Passed	tests/test_integridad.py::test_integridad	249 ms	

ID	Test (archivo)	Propósito breve	Resultado	Duración (ms)
1	`tests/test_avalancha.py`	Medir efecto avalancha (cambio 1 bit en entrada)	Passed — 1 bit distinto de 12480	414
2	`tests/test_cli_encrypt.py`	Validar CLI: cifrar/descifrar con parámetros	Passed	301
3	`tests/test_entropy.py`	Medir entropía Shannon del ciphertext	Passed — 7.8541 bits/byte	243
4	`tests/test_feistel_block.py`	Verifica ciclo bloque (encrypt→decrypt = original)	Passed	16
5	`tests/test_feistel_ctr.py`	Verifica CTR (reversibilidad, long text)	Passed	38

6	`tests/test_integrida d.py`	Integridad básica (cifrar→descifrar)	Passed	249

2. Conclusiones

Integridad y reversibilidad: El cifrador cumple su objetivo básico: `encrypt -> decrypt` devuelve el mensaje original. Esto valida la implementación Feistel (diseño correcto de rondas/inversa).

1. **Entropía alta**. Con ~7.85 bits/byte, los ciphertexts muestran una buena aleatoriedad estadística — indicador positivo para confidencialidad frente a análisis estadístico simple.

2. **Avalancha extremadamente baja**. 1 bit distinto de 12,480 bits → tasa ≈ 0.008%.

    - **Problema serio**. Un cifrador bien diseñado debe propagar cambios pequeños en el plaintext a ~50% de los bits del ciphertext.

    - **Posibles causas**.

        - Modo CTR + misma nonce/keystream mal aplicado al comparar (si keystream es idéntico y sólo el bloque modificado coincide, la diferencia puede concentrarse).

        - Función de ronda no suficientemente no-lineal o pocas rondas (ROUNDS insuficiente).

        - Comparación del ciphertext realizada en forma que ignora zonas aleatorias (ej. usando distinto nonce) — la metodología de medición podría estar equivocada.

    - **Implicación**. El cifrador puede ser vulnerable a ataques diferenciales y no proporcionar difusión suficiente.

3. **Performance**. Los tiempos de test son bajos (ms) para mensajes de prueba; la implementación es razonablemente rápida para archivos pequeños

Ataques viables

Con los resultados actuales (alta entropía pero baja avalancha) los ataques más probables son:

- **Ataque diferencial**
    - Si un cambio de 1 bit produce cambios pequeños en el ciphertext, es viable construir diferencias en la entrada que provoquen patrones en la salida y recuperar subclaves parcialmente.
    - Riesgo alto si la F-function es lineal o las rondas son pocas.

- **Ataque por texto elegido**
    - Un atacante que pueda cifrar mensajes escogidos y observar ciphertexts puede explotar la falta de difusión para deducir relaciones internas.

- **Fuerza bruta sobre clave derivada de contraseña**
    - Si se permite la derivación desde contraseña débil, se pueden realizar ataques de diccionario. Mitigación: PBKDF2 con iteraciones altas (ya usas 200k), usar contraseñas fuertes o claves aleatorias.

- **Manipulación y falta de autenticación**
    - CTR no provee integridad. Un atacante puede flipar bits en ciphertext y generar cambios controlados en plaintext tras descifrado (bit-flipping). Si no tienes MAC/HMAC, integridad/ autenticación están

comprometidas.

- **Reutilización de nonce**
    - Si nonce se reutiliza para dos mensajes con la misma clave, se filtra información (XOR of plaintexts).

3. Recomendaciones

1. **Añadir autenticación (obligatorio)**
    - Añadir HMAC-SHA256 sobre `header||nonce||ciphertext` o usar un modo autenticado (AES-GCM). Esto protege integridad y autenticidad.
2. **Corregir / profundizar test de avalancha**
    - Ejecutar el test de avalancha en muchas muestras aleatorias (N≥1000), usando el mismo nonce y luego distinto nonce para entender comportamiento, y calcular porcentaje medio de bits distintos. Guardar CSV con tasas.
    - Asegurarse que el método de comparación compare keystreams equivalentes; medir difusión por bloque y global.
3. **Fortalecer F-function**
    - Introducir S-box no-lineal (bien diseñada) y permutación de bytes/rotaciones; o usar AES como F (si se permite).

    - Incrementar número de rondas (ej. 16, 24) hasta que avalancha se acerque a 50%.

4. **Evitar errores en CTR**

    - No usar nonce aleatorio sin registrarlo en el header (tu header ya guarda nonce, OK). Evitar reutilización de nonce con la misma clave.

5. **Mejorar key handling**

    - Si permites contraseña, forzar verificaciones de entropía mínima; guardar `key.bin` sólo si es generado aleatoriamente. Ofrecer opción de exportar clave con protección (escrow).

6. **Mediciones más exhaustivas**

    - Medir avalancha por bit (distribución) y no solo un único experimento.
    - Hacer curvas de performance para 1 KB, 10 KB, 100 KB y 1 MB.

Resultados esperados después de las mejoras

- Avalancha: tasa media cercana al 50% (±5%).
- Entropía: ~8 bits/byte.
- Integridad: detecta cualquier manipulación (HMAC o AEAD).
- Performance: aceptable; si agregas HMAC y/o más rondas, medir el impacto y justificar.