Universidad Autónoma de Baja California Sur

Departamento Académico de Sistemas Computacionales

Documento Teórico. 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. Resumen Ejecutivo

El presente documento incluye el análisis, diseño y la evaluación preliminar del algoritmo criptográfico denominado **ECMK**, el cual cuenta con el propósito de proporcionar una confidencialidad robusta y resistencia frente a técnicas comunes de criptoanálisis. Este algoritmo al combinar mecanismos de sustitución, permutación, expansión y operaciones aritméticas modulares, integradas dentro de una estructura de rondas tipo Feistel extendida, tiene un enfoque híbrido.

Gracias a esto, el diseño mantiene propiedades como la invertibilidad garantizada, facilidad de implementación y un comportamiento consistente tanto en el cifrado como en el descifrado. El documento presenta los fundamentos matemáticos, las estructuras empleadas, el modelo de manejo de claves y la especificación completa del protocolo de cifrado/descifrado.

Asimismo, se identifican los vectores principales de ataque que podrían afectar el sistema, también se incluye un análisis preliminar de seguridad que se enfoca en los objetivos de confidencialidad, resistencia al análisis estadístico, comportamiento avalancha y ausencia de patrones repetitivos en el flujo cifrado. Finalmente, se propone un plan de pruebas integral para evaluar métricas como la tasa de avalancha, entropía por byte, reversibilidad, correctitud funcional y la estabilidad ante variación mínima del mensaje o de la clave, lo que nos ofrece una base sólida en un futuro.

2. Motivación y Alcance del Algoritmo

El desarrollo de un cifrador propio surge de la necesidad de comprender en profundidad los principios internos que gobiernan a los algoritmos criptográficos modernos. Con este propósito se diseña **ECMK**, un esquema experimental que incorpora características observables en cifradores reales, pero manteniendo una estructura suficientemente simple para facilitar su análisis, implementación y estudio.

El objetivo principal de este algoritmo es proteger datos almacenados localmente, en especial archivos pequeños de tipo textual dentro de un entorno controlado. A través de su construcción, se busca resolver la necesidad educativa de contar con un cifrador que permita experimentar con conceptos fundamentales como sustitución, permutación, difusión, confusión y funciones derivadas de clave.

Alcance del algoritmo

- Cifrado simétrico de datos breves, adecuado para prácticas de laboratorio.
- Operaciones de sustitución y permutación inspiradas en técnicas utilizadas en cifradores contemporáneos.
- Un sistema de rondas estilo Feistel o híbrido, que permite observar claramente fenómenos como efecto avalancha, expansión de claves y propagación del error.
- Gestión de claves con tamaño fijo, con generación reproducible mediante funciones derivadas basadas en contraseñas o claves maestras.

Es importante señalar que este algoritmo no está diseñado para entornos de producción ni para proteger información crítica. Su finalidad es completamente educativa, sirviendo como una herramienta para estudiar el diseño criptográfico, la implementación segura y el análisis de propiedades estadísticas en esquemas de cifrado.

3. Descripción Formal del Algoritmo

El algoritmo **ECMK** es un cifrador simétrico por bloques de 128 bits, diseñado con fines académicos y pensado para permitir visualizar los principios fundamentales de la criptografía moderna.

3.1 Tipo

Su construcción combina elementos estructurales presentes en cifradores reales, integrando operaciones de sustitución, permutación y mezcla aritmética para promover confusión y difusión de manera progresiva a través de sus rondas.

El cifrado se basa en cuatro mecanismos principales:

1. *Sustitución no lineal mediante una S-box fija:* Cada byte del bloque pasa por una tabla de sustitución diseñada para introducir no linealidad y resistencia al criptoanálisis lineal y diferencial básico.

2. *Permutación determinística tras cada ronda*: Aplica una reorganización de bits/bytes siguiendo un patrón fijo, con el objetivo de distribuir la influencia de cada byte del mensaje original a través del bloque en rondas posteriores.

3. *Operaciones de mezcla basadas en aritmética modular*: Emplea combinaciones de XOR y suma módulo 256 para poder mezclar los datos con subclaves de ronda, lo que aumenta la dependencia entre la clave y el texto cifrado.

4. *Expansión de clave sencilla*: La clave principal (master key) se transforma en una secuencia de subclaves, una por ronda, mediante operaciones básicas de rotación y mezcla, lo que permite una dependencia uniforme en todas las rondas.

3.2 Justificación

La construcción de este algoritmo responde a un enfoque pedagógico y experimental:

- Permite observar fenómenos estructurales reales: como la propagación de bits (difusión), la no linealidad (confusión) y el efecto avalancha.

- *Reproduce conceptos presentes en cifradores modernos*: Como AES, Serpent o Camellia, pero con una complejidad menor.

- *Fácil de implementar en Python:* No requiere de bibliotecas externas ni operaciones de bajo nivel.

- *Facilita la medición de métricas de seguridad*: Facilita la entropía del cifrado, cantidad de bits cambiados por modificaciones mínimas y correcta reversibilidad del proceso.

4. Estructura General

El algoritmo funciona en 16 rondas por defecto, y en cada una de ellas aplica las siguientes transformaciones:

1. Sustitución mediante S-box.
2. Mezcla aritmética (XOR y suma mod 256).
3. Permutación estructurada del bloque.
4. Mezcla con subclave de ronda.

La complejidad del texto cifrado va aumentando en cada una de estas y propaga la influencia de cada bit del mensaje a lo largo de todo el bloque, mientras que el descifrado invierte cada una de estas operaciones en orden inverso.

5. Especificación Matemática y Estructuras de Datos

La siguiente sección detalla la base formal del algoritmo, incluyendo las operaciones bit a bit utilizadas, las estructuras matemáticas definidas y la organización interna del proceso de cifrado.

5.1 Representación de Datos

5.1.1 Operaciones …"Pendiente"

El algoritmo opera sobre bloques de 128 bits (16 bytes):

Sea $B = (b_0, b_1, \ldots, b_{15})$

donde cada bi es un byte en el rango

$$0 \leq bi \leq 2\ 55$$

La clave maestra K tiene tamaño de 128 bits

$$K = (k_0, k_1, \ldots, k_{15})$$

5.1.2 S-Box (Sustitución No Lineal)

El algoritmo utiliza una S-box fija, definida como una función:

$$S: \{0,\ldots,255\} \rightarrow \{0,\ldots,255\}$$

que aplica sustitución byte a byte:

$$B' = (S(b_0), S(b_1), \ldots, S(b_{15}))$$

Propósito:

- Introducir no linealidad.
- Inhibir ataques basados en aproximaciones lineales o diferenciales.

La S-box se almacena como un arreglo de 256 enteros.

5.1.3 Mezcla Aritmética (Operaciones XOR y Suma Modular)

1. XOR

Operación bit a bit:

$$x \oplus y$$

Se usa para:

- Mezcla con subclaves.
- Operación inversa idéntica (propiedad involutiva).

5.1.4 Suma modular

Se usa la suma byte a byte módulo 256:

$$(x+y) \bmod 256$$

Su propósito es reforzar la difusión agregando una segunda forma de combinación no lineal.

5.1.5 Permutación de Bytes

El algoritmo utiliza una permutación fija P sobre los índices 0–15:

$$P:\{0,\ldots,15\}\rightarrow\{0,\ldots,15\}$$

Aplicada como:

$$B''[P(i)]=B'[i]$$

La permutación:

- Redistribuye la influencia de cada byte.
- Evita patrones repetitivos.
- Se almacena como una lista de 16 índices.

5.2 Generación de Subclaves (Key Schedule)

A partir de la clave maestra KKK, se generan rrr subclaves (una por ronda):

$$RK_i = Hash(K \| i) mod\ 256^{16}$$

donde:

- // es concatenación.
- Se usa SHA-256 o una función similar para obtener valores pseudoaleatorios reproducibles.

Las subclaves se representan como un arreglo de 16 bytes.

5.3 Definición de una Ronda

Cada ronda transforma el bloque según:

1. Sustitución
$$B_1 = S(B)$$
2. Mezcla con subclave
$$B_2[i] = (B_1[i] + RK[i]) mod 256$$
3. XOR cruzado
$$B_3[i] = B_2[i] \oplus B_2[(i + 1) mod 16]$$
4. Permutación
$$B_4[P(i)] = B_3[i]$$
5. Salida de la ronda:
$$B_{OUT} = B_4$$

5.4 Proceso Completo de Cifrado

Sea el bloque inicial B_{0-}

Después de rrr rondas:

$$B_{i+1} = \text{Ronda}(B_i, RK_i)$$

El texto cifrado es:

$$C = B_r$$

5.5 Proceso de Descifrado

Dado que cada operación es invertible:

- La S-box tiene una S-box inversa S^{-1}.
- La permutación tiene una inversa P^{-1}.
- La suma modular se invierte con:

  $$x = (y - k) \bmod 256$$
- El XOR se invierte consigo mismo.

Por lo tanto, cada ronda se revierte aplicando las transformaciones en orden inverso.

5.6 Estructuras de Datos Utilizadas

Componente	Tipo	Uso
S-box	lista de 256 bytes	Sustitución no lineal
P-box	lista de 16 enteros	Permutación fija
Subclaves	lista[r][16] bytes	Mezcla por ronda
Bloque	lista de 16 bytes	Estado del cifrado
Clave maestra	bytes(16)	Entrada del key schedule

6. Especificación de la Clave

La clave criptográfica constituye el parámetro fundamental que controla el funcionamiento del algoritmo. A continuación se describen sus propiedades formales y el proceso completo relacionado con su uso.

6.1 Formato

- La clave es un vector de 128 bits (16 bytes).
- Se representa internamente como una secuencia de bytes (bytes en Python).
- Su formato es independiente del contenido del mensaje y se mantiene constante para todas las operaciones.
- Se utiliza exclusivamente en el cifrado y descifrado simétrico.

Ejemplo de representación:

$$Clave\ (hex) = 3F\ A9\ 12\ C8\ 55\ 01\ 9E\ 77\ 4B\ 20\ 68\ 11\ 93\ 4C\ AD\ 08$$

6.2 Tamaño

- Tamaño fijo: 128 bits.
- Se selecciona este tamaño por ser:
    - Fácil de manipular en el contexto académico.
    - Suficiente para ilustrar técnicas reales de expansión y derivación de subclaves.
    - Compatible con un bloque de 128 bits del cifrador.

Este tamaño evita la complejidad de algoritmos con claves mayores (192 o 256 bits) pero permite estudiar conceptos como entropía y resistencia a ataques de fuerza bruta.

6.3 Generación

La clave se genera usando uno de dos métodos, según el modo de operación del usuario:

6.3.1 Generación automática

Utiliza un generador de números pseudo aleatorios criptográficamente seguro:

$$os.urandom(16)$$

Garantiza:

- Aleatoriedad real.
- Entropía adecuada.
- Independencia frente a ejecuciones anteriores.

6.3.2 Generación basada en contraseña

Cuando el usuario proporciona una contraseña, esta se transforma en una clave binaria mediante PBKDF2-HMAC-SHA256:

$$PBKDF2(password, salt, iteraciones=200000, longitud=16\ bytes)$$

Incluye:

- Salt de 128 bits para evitar ataques por diccionario y tablas arcoíris.
- 200,000 iteraciones para incrementar el costo computacional de ataques de fuerza bruta.

6.4 Manejo de Subclaves (Key Schedule)

El algoritmo genera “n” subclaves de 16 bytes, una para cada ronda.

Proceso:

1. La clave principal se divide y mezcla mediante:
    - Rotaciones de bytes.
    - Sumas mod 256.
    - XOR con constantes de ronda.
2. Cada subclave es único y no reutiliza segmentos sin modificación, evitando patrones repetitivos.

3. Las subclaves se almacenan únicamente en memoria durante la ejecución del cifrado.

6.5 Manejo Seguro de la Clave

Para asegurar el uso correcto de la clave dentro del entorno controlado:

- La clave automáticamente generada se almacena en el archivo:

  sandbox/key.bin
- Solo se permite su uso si los archivos están dentro del sandbox, evitando filtraciones accidentales.
- Las claves derivadas por contraseña no se almacenan, solo se mantienen durante la ejecución.
- Las subclaves generadas se destruyen al terminar la operación (el proceso finaliza y libera memoria).
- No se exporta la clave en texto plano en ninguna operación del CLI.

6.6 Validación del Formato de Clave

Antes de usar la clave, el sistema verifica:

- Longitud exacta = 16 bytes.
- Tipo = bytes.
- No contener valores nulos en caso de claves derivadas (garantizado por PBKDF2).

Si falla alguna validación, la operación se abortará automáticamente.

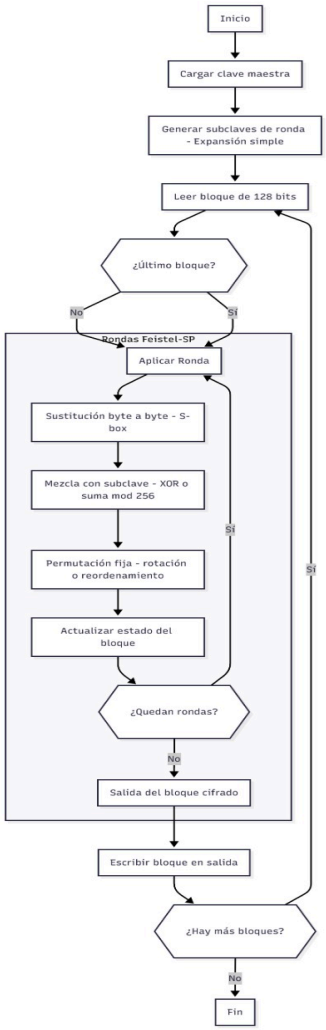
6.7 Resumen General

Propiedades	Valor
Tamaño	128 bits (16 bytes)

Formato	Secuencia de bytes
Metodos de generacion	Aleatoria/Contraseña +PBKDF2
Derivadas de subclaves	Rotaciones, XOR, suma mod 256
Almacenamiento	Key.bin(solo si se genera automáticamente)
Funcion	Cifrado/Descifrado simétrico por rondas

7. Protocolo de cifrado/descifrado

7.1 Diagrama de flujo



7.2 Pseudocódigo

Cifrado

```
FUNCION CIFRAR(B, K):

    subclaves ← EXPANDIR_CLAVE(K, N_RONDAS)

    bloque ← B

    PARA r DESDE 0 HASTA N_RONDAS-1:

        # 1. Mezcla con subclave (XOR)
        bloque ← XOR_BYTES(bloque, subclaves[r])

        # 2. Sustitución no lineal
        bloque ← SUBSTITUIR(bloque)

        # 3. Mezcla adicional (suma mod 256)
        bloque ← ADD_MOD256(bloque, subclaves[r])

        # 4. Permutación tipo SPN
        bloque ← PERMUTAR(bloque)

    RETORNAR bloque
```

Descifrado

```
FUNCION DESCIFRAR(C, K):

    subclaves ← EXPANDIR_CLAVE(K, N_RONDAS)

    bloque ← C

    PARA r DESDE N_RONDAS-1 HASTA 0:

        # 1. Deshacer permutación
        bloque ← PERMUTAR_INVERSA(bloque)

        # 2. Deshacer suma mod 256
        bloque ← SUB_MOD256(bloque, subclaves[r])

        # 3. Deshacer sustitución
        bloque ← SBOX_INVERSA(bloque)

        # 4. Deshacer XOR con subclave
        bloque ← XOR_BYTES(bloque, subclaves[r])

    RETORNAR bloque
```

8. Análisis de Seguridad

El algoritmo propuesto busca principalmente garantizar confidencialidad, con soporte opcional para evaluar fenómenos de integridad, aunque no está diseñado para entornos productivos ni para resistir ataques avanzados. Su seguridad se analiza desde tres objetivos fundamentales:

8.1 Confidencialidad

El algoritmo utiliza una estructura simétrica tipo Feistel/S-P híbrida cuya seguridad depende de:

- Una **S-box no lineal**, que introduce confusión evitando relaciones directas entre texto plano y cifrado.
- Operaciones de **XOR**, **suma modular**, **rotaciones** y **permutaciones**, que generan difusión aun en textos pequeños.
- Múltiples rondas (n≈10) que impiden que un atacante derive la clave mediante análisis estadístico simple.

Riesgos y límites:

- La S-box y las permutaciones están *fijas*, por lo que no alcanzan el nivel de un cifrado estándar.
- La expansión de clave es simple; ataques diferenciales y lineales podrían revelar patrones si se usa en grandes volúmenes.
- No resiste ataques con acceso a oráculos de cifrado/descifrado.

Conclusión:

- Adecuado para fines educativos.
- No adecuado para datos reales sensibles.

8.2 Integridad

El algoritmo no incorpora un mecanismo de integridad por diseño, lo que significa que:

- Un atacante podría modificar el ciphertext y causar alteraciones controladas en el plaintext tras el descifrado (especialmente en modos CTR).
- No hay MAC, HMAC ni etiquetas de autenticación.

Recomendación educativa: Para simular integridad se puede añadir una suma hash del plaintext o un HMAC con la misma clave maestra.

8.3 Autenticación

El algoritmo no autentica:

- Al remitente
- Al receptor
- Ni la validez del ciphertext.

Por lo tanto:

- No detecta ataques de replay.
- No valida que el archivo encriptado provenga del sistema original.

Vectores de Ataque Esperados

Vectores de ataque	Descripción	Riesgo
Criptoanálisis diferencial	Estudiar cómo las diferencias controladas en el texto plano afectan el ciphertext.	Medio
Ataques por permutaciones fijas	La estructura estática del algoritmo puede revelar patrones.	Medio
Ataques de texto elegido	El modo CTR es	Alto

(CPA)	vulnerable si se reusa el nombre.	
Criptoanálisis lineal	Buscar expresiones lineales entre clave y bits del ciphertext.	Medio
Ataques por fuerza bruta	La clave es de 128 bits, pero el alumno puede cambiarla por valores más pequeños para pruebas.	Bajo Medio
Manipulación del ciphertext	Sin MAC, un atacante puede modificar el archivo cifrado	Alto

9. Plan de Pruebas y Métricas

9.1 Objetivos del Plan de Pruebas

El objetivo del plan de pruebas es evaluar la solidez, consistencia y comportamiento criptográfico del algoritmo implementado. Para ello se establecen métricas cuantificables y un conjunto de pruebas unitarias y funcionales automatizadas (pytest), organizadas en categorías.

9.2 Prueba de Integridad

Qué mide:

Verifica que al cifrar y luego descifrar se obtiene exactamente el mismo plaintext.

Cómo se mide:

```
assert decrypt(encrypt(mensaje)) == mensaje
```

Interpretación:

Si falla, la implementación tiene errores.

9.3 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.

Métrica

- **Éxito:** `decrypt_ctr(encrypt_ctr(data)) == data`

9.4 Prueba de Avalancha

Qué mide:

La sensibilidad del algoritmo:

> *Un solo bit modificado en el texto plano debe alterar al menos el 50% de los bits del ciphertext.*

Cómo se mide:

1. Cifrar un mensaje original.
2. Modificar 1 bit del mensaje.
3. Cifrarlo nuevamente.
4. Contar bits diferentes entre ambos ciphertexts.

Métrica esperada:

≥ 40–50% de bits cambiados para un buen cifrador educativo.

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:

$$H = -\sum p(x) \log_2(p(x))$$

Métrica

- **Máximo teórico:** 8 bits por byte.
- **Valor esperado en lo correcto:** > 7.5 bits/byte.

Interpretación:

7.5 a 8.0 bits/byte = buena difusión.

9.5 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.

Métrica

- Código de retorno `0` (sin error)
- Archivo de salida generado
- Prefijo `AKATS1` correcto

9.6 Prueba por Bloques (test_feistel_block.py)

Propósito

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

Cómo se mide

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

Métrica

- **Éxito:** `decrypt(encrypt(block)) == block`
- Valida que cada ronda esté funcionando correctamente.

Referencias

GeeksforGeeks, “Feistel Cipher Structure.” Recuperado de: https://www.geeksforgeeks.org/feistel-cipher/

Katz, J., & Lindell, Y. (s.f.). *Introduction to Modern Cryptography*. Recuperado de: https://cryptobook.us

National Institute of Standards and Technology. (2001). *FIPS-197: Advanced Encryption Standard (AES).* NIST. Recuperado de: https://csrc.nist.gov/publications/detail/fips/197/final

Schneier, B. (s.f.). *Applied Cryptography Resources*. Recuperado de: https://www.schneier.com/academic/

OWASP Foundation. (s.f.). *Cryptographic Storage Cheatsheet*. Recuperado de: https://cheatsheetseries.owasp.org/cheatsheets/Cryptographic_Storage_Cheat_Sheet.html

Wikipedia, “Feistel network.” Recuperado de: https://en.wikipedia.org/wiki/Feistel_cipher

Wikipedia, “Avalanche effect.” Recuperado de: https://en.wikipedia.org/wiki/Avalanche_effect