

Primer Parcial - Takehome

1st Maria Gabriela Espinosa, 2nd María Fernanda Ocampo, 3rd Luis Gabriel Guayambuco
1202419, 1202412, 1201505

Seguridad Informática
Universidad Militar Nueva Granada
Bogotá, Colombia

Abstract—This work aims to present the practical implementation and theoretical analysis of classical cryptographic algorithms (Caesar, Vigenère), modern algorithms (Fernet/AES), hash functions (SHA-256, HMAC), salt authentication, RSA digital signatures, basic blockchain simulation, and secure communications (VPN/HTTPS). Practical development in Python is combined with theoretical analysis. Fundamentals such as randomness (RNG vs. PRNG), attack models (COA, KPA, CPA, CCA), Kerckhoff's principle, computational security, and ECB risks are explored, contrasting cryptography with steganography and obfuscation. Real-world applications are analyzed according to the MinTIC Security Model and NIST/ISO standards. The results demonstrate hash irreversibility, blockchain immutability, and the benefits of secure channels, contributing to a comprehensive understanding of cybersecurity in educational and business contexts.

Index Terms—Criptografía, Firma Digital, Blockchain, Principios de Kerckhoff, Esteganografía, Seguridad Informática, Hash, VPN, HTTPS

I. INTRODUCCIÓN

La criptografía es un pilar fundamental en la seguridad informática porque proporciona las herramientas matemáticas y computacionales necesarias para proteger la confidencialidad, integridad, autenticidad y no repudio de la información en entornos digitales. Su importancia radica en que, sin ella, las comunicaciones electrónicas, las transacciones financieras, el almacenamiento de datos sensibles y la identidad digital serían vulnerables a ataques de interceptación, modificación o suplantación. En un mundo cada vez más interconectado, donde los datos viajan por redes abiertas y se procesan en sistemas distribuidos, la criptografía actúa como la última línea de defensa contra amenazas cibernéticas.[1].

Desde una perspectiva histórica, los sistemas criptográficos evolucionaron desde los cifrados clásicos, como el cifrado César (sustitución monoalfabética) y el cifrado Vigenère (polialfabético), con la necesidad constante de adaptarse a nuevos desafíos. Sin embargo, estos sistemas presentan vulnerabilidades frente a técnicas de criptoanálisis estadístico, particularmente el análisis de frecuencia [2]. La necesidad de resistencia frente a modelos de ataque más sofisticados impulsó el desarrollo de la criptografía moderna.

La transición hacia la criptografía moderna se apoyó en disciplinas como la teoría de números, el álgebra abstracta y la complejidad computacional, dando lugar a algoritmos robustos como el Advanced Encryption Standard (AES) y el Secure Hash Standard (SHS), estandarizados por organismos como NIST e ISO. Estos algoritmos resisten ataques de fuerza bruta y criptoanálisis avanzado gracias a fundamentos matemáticos sólidos.

Organismos internacionales como la International Organization for Standardization (ISO) y el National Institute of Standards and Technology (NIST) han definido estándares formales que regulan el uso de algoritmos criptográficos en sistemas de información, tales como el Advanced Encryption Standard (AES) y el Secure Hash Standard (SHS) [3], [4].

A partir de estos cimientos, se construyen aplicaciones esenciales para la seguridad actual:

- 1) **Funciones hash criptográficas (como SHA-256):** Transforman cualquier mensaje en un resumen de longitud fija, garantizando integridad y autenticidad. Son la base de la verificación de datos, almacenamiento seguro de contraseñas y generación de firmas digitales.
- 2) **HMAC (Hash-based Message Authentication Code):** Combina una función hash con una clave secreta, proporcionando autenticidad e integridad en protocolos como TLS, esencial para la seguridad en la web.[5].
- 3) **RSA y Firma Digital:** El algoritmo asimétrico RSA permite cifrar y firmar mensajes. La firma digital asegura que un documento proviene de quien dice ser y no ha sido alterado, logrando no repudio mediante el uso de claves pública y privada.
- 4) **Blockchain:** Aprovecha el encadenamiento criptográfico de bloques mediante hashes, logrando inmutabilidad y transparencia en sistemas descentralizados como Bitcoin. La seguridad de blockchain depende de la resistencia a colisiones de las funciones hash y de mecanismos de consenso distribuido.[6]
- 5) **TLS/HTTPS y VPN:** estos protocolos de comunicación

segura utilizan criptografía asimétrica para el intercambio inicial de claves y cifrado simétrico para la sesión, protegiendo la privacidad e integridad de los datos transmitidos en internet.

Esta cadena conceptual (desde los cifrados clásicos hasta las tecnologías modernas) demuestra cómo la criptografía ha evolucionado para responder a las necesidades de seguridad en cada época, y cómo sus principios fundamentales se entrelazan para construir sistemas complejos y confiables.

El propósito del presente documento es integrar estos fundamentos teóricos con una implementación práctica en Python, permitiendo analizar tanto la base matemática de los algoritmos criptográficos como sus implicaciones de seguridad en aplicaciones reales. A través del estudio de funciones hash, HMAC, RSA, blockchain y protocolos como HTTPS y VPN, se busca comprender no solo cómo funcionan internamente, sino también cómo su correcta aplicación garantiza la protección de la información en entornos digitales contemporáneos.

II. FUNDAMENTOS TEÓRICOS

A. Aleatoriedad y Generadores (RNG vs PRNG)

La aleatoriedad es un componente estructural en criptografía moderna, ya que garantiza imprevisibilidad en la generación de claves, vectores de inicialización, nonces y sales criptográficas. Si los valores generados presentan patrones deterministas o correlaciones explotables, la seguridad del sistema se reduce drásticamente, incluso cuando el algoritmo subyacente es matemáticamente robusto [1].

En términos formales, un sistema criptográfico es tan fuerte como la entropía de sus parámetros secretos. La entropía mide la incertidumbre asociada a una variable aleatoria; una baja entropía implica espacio de búsqueda reducido y facilita ataques por fuerza bruta.

Existen dos categorías principales:

- 1) **RNG (Random Number Generator):** Generadores de números verdaderamente aleatorios. Se basan en fenómenos físicos impredecibles (ruido térmico, fluctuaciones cuánticas). Son utilizados en hardware criptográfico o sistemas operativos modernos mediante fuentes de entropía del sistema.
- 2) **PRNG (Pseudo-Random Number Generator):** Algoritmos deterministas que producen secuencias aparentemente aleatorias a partir de una semilla inicial. Si la semilla es conocida, la secuencia es completamente predecible. Son adecuados para simulación, pero no todos son seguros criptográficamente.

En Python, el módulo `random` implementa un PRNG basado en Mersenne Twister. No es criptográficamente seguro. El

módulo `secrets` y `os.urandom` acceden a fuentes de entropía seguras del sistema y son apropiados para aplicaciones criptográficas.

B. Modelos de Ataque (COA, KPA, CPA, CCA)

La seguridad criptográfica se evalúa bajo modelos de ataque formales[7]:

- 1) **COA (Ciphertext-Only Attack):** o Ataque solo con texto cifrado: El atacante únicamente dispone del texto cifrado. Es el escenario más débil y el más común en la práctica. El análisis de frecuencia sobre cifrados clásicos es un ejemplo típico.
- 2) **KPA (Known Plaintext Attack):** o Ataque con texto claro conocido: El adversario posee pares de texto claro y su correspondiente texto cifrado. Puede explotarse en cifrados por sustitución si se dispone de suficiente material.
- 3) **CPA (Chosen Plaintext Attack):** o Ataque con texto claro elegido: El atacante puede elegir textos claros arbitrarios y obtener sus cifrados. Es relevante en cifrados de flujo y modos de operación débiles como ECB.
- 4) **CCA (Chosen Ciphertext Attack):** o Ataque con texto cifrado elegido: El adversario puede descifrar textos cifrados elegidos (excepto el objetivo). Fundamental para evaluar esquemas de cifrado asimétrico y modos autenticados.

Los cifrados clásicos como César son vulnerables incluso bajo COA, debido al análisis de frecuencia. Este método explota la distribución estadística de letras en un idioma. En español, por ejemplo, la letra "E" es la más frecuente. Si en un texto cifrado una letra domina estadísticamente, puede inferirse el desplazamiento aplicado.

En Vigenère, el análisis se complica al usar múltiples alfabetos, pero puede romperse mediante el método de Kasiski o análisis de coincidencia de índices[2]. Estos ataques demuestran que seguridad basada solo en sustitución alfabética carece de resistencia frente a adversarios con conocimiento estadístico.

C. Principio de Kerckhoff

El principio de Kerckhoff establece que un sistema criptográfico debe ser seguro incluso si todo el algoritmo es público, excepto la clave[8]. Este principio es fundamental porque elimina la dependencia de seguridad por ocultamiento y obliga a fundamentar la seguridad en propiedades matemáticas verificables.[9]

D. Seguridad Computacional y Tamaño de Llaves

La seguridad computacional implica que romper un sistema requiere recursos de tiempo y procesamiento imprácticos con

la tecnología disponible[1]. No se afirma que el sistema sea irrompible en términos absolutos, sino que el costo excede cualquier beneficio razonable.

La relación con el tamaño de llave es directa:

- 1) **AES-128:** Espacio de búsqueda de

$$2^{128}$$

posibles claves.

- 2) **RSA-2048:** seguridad basada en la dificultad de factorizar enteros de 2048 bits.

- 3) **SHA-256:** resistencia a colisiones =

$$2^{128}$$

operaciones (ataque birthday).

Un incremento lineal en tamaño de clave produce crecimiento exponencial en el espacio de búsqueda. Por ello, duplicar la longitud de clave no duplica la seguridad; la multiplica exponencialmente.

El modo ECB (Electronic Codebook) no es recomendado porque cifra bloques idénticos de manera idéntica. En imágenes, esto preserva patrones visuales. El ejemplo clásico es la imagen del pingüino Tux de Linux cifrada con AES-ECB: aunque los datos están cifrados, la silueta del pingüino sigue siendo visible porque los bloques repetidos generan salidas repetidas[10]. Esto demuestra que confidencialidad no depende solo del algoritmo, sino del modo de operación.[11]

E. Criptografía vs Esteganografía vs Ofuscación

Criptografía transforma información para que sea ininteligible sin la clave. Ejemplo: cifrar un archivo con AES.

Esteganografía oculta la existencia del mensaje, por ejemplo, insertando datos en los bits menos significativos de una imagen.

Ofuscación modifica código o datos para dificultar su comprensión, como renombrar variables en software.

La seguridad por ocultamiento es considerada una vulnerabilidad porque depende del secreto del mecanismo, no de la fortaleza matemática. Si el método se descubre, el sistema colapsa. Este enfoque contradice el principio de Kerckhoff[12][13].

III. METODOLOGÍA

La metodología consistió en implementar un modular de cada ejercicio en scripts independientes, para así identificar cada uno de los puntos en concreto, con funciones claramente definidas y muy bien comentareadas.

Se realizaron pruebas unitarias con casos controlados, es decir: para mensajes conocidos para César, archivos de prueba

para hash, y así sucesivamente para cada uno. También se hizo la validación de los resultados mediante Google Colab como editor principal.

- 1) **Entorno de desarrollo:** El desarrollo del taller se realizó en Python 3, utilizando las siguientes librerías:

```
1 # Ejecuta para instalar las
2 dependencias
3 !pip install pycryptodome Pillow -q
4 print(Librerías instaladas
correctamente.)
```

```
1 # Instalación necesaria (ejecutar
2 una vez):
3 # !pip install pycryptodome Pillow -
4 q
5 # Lo cual importara las siguientes
6 librerías:
7
8 import hashlib
9 import hmac
10 import os
11 import datetime
12 import urllib.request
13 import ssl
14 import binascii
15 from http.client import HTTPResponse
16
17 from Crypto.Cipher import AES,
18 PKCS1_OAEP
19 from Crypto.PublicKey import RSA
20 from Crypto.Signature import
21 pkcs1_15
22 from Crypto.Hash import SHA256
23 from Crypto.Random import
24 get_random_bytes
25 from Crypto.Util.Padding import pad,
26 unpad
27 from PIL import Image
```

- **hashlib:** Para funciones hash SHA-256 y derivación HMAC.

Permite trabajar con algoritmos seguros y funciones de resumen de mensajes, es decir, que puede transformar datos, como por ejemplo contraseñas o archivos, en una representación única, compacta e irreversible. Si bien algunos de estos como son MD5 y SHA-1 tienen vulnerabilidades conocidas, siguen siendo útiles en aplicaciones no críticas, como por ejemplo para generar checksums. [14]

- **cryptography:** Proporciona recetas criptográficas con interfaces de alto nivel (para tareas criptográficas comunes, como cifrado simétrico, cifrados simétricos, resúmenes de mensajes y funciones de derivación de claves (cifrado simétrico Fernet (AES) y operaciones asimétricas RSA (generación de claves, firma y verificación)). Las recetas de cifrado simétrico de alto nivel ayudan a implementar algoritmos complejos de manera rápida y sencilla.) y bajo nivel (para algoritmos criptográficos, lo que permite un control y una personalización más gran-


```

9         resultado += chr((ord(char) - ord('A'
10    ) - desplazamiento) % 26 + ord('A'))
11         idx_clave += 1
12     else:
13         resultado += char
14     return resultado

```

Se realiza la prueba del cifrado con un mensaje, el cual fue "El profe es super genial uwu", y la clave usada fue "VIGENERE" se cifra el mensaje con:

```

1 cifrado_vig = cifrar_vigenere(mensaje,
    clave_vigenere)

```

Para descifrar y comprobar que recuperamos el original (en mayúsculas, porque así se convirtió) se hace de la siguiente forma:

```

1 print(fDescifrado: {descifrar_vigenere(
    cifrado_vig, clave_vigenere)})

```

Con lo cual nos da el siguiente resultado:

```

=== Cifrado Vigenère ===
Original: El profe es super genial uwu
Cifrado con clave 'VIGENERE': ZT VVB JV IN AATRV XIIQGP HAL
Descifrado: EL PROFE ES SUPER GENIAL UWU

```

Fig. 1. Ataque por fuerza bruta

3) *Ataque por fuerza bruta:* Se realiza la prueba con el siguiente código:

```

1 print(\n=== Para la Frase===)
2 print(\nAtaque de fuerza bruta sobre 'Ov dsowzy
   oc vy aeo nodobwsxk vk coqebnskn. Myx dsowzy
   cepmssoxd xknk oc sxrkmuklvo':)
3 fuerza_bruta_cesar(Ov dsowzy oc vy aeo nodobwsxk
   vk coqebnskn. Myx dsowzy cepmssoxd xknk oc
   sxrkmuklvo)
4 print(\n=== Para el Nombre===)
5 print(\nAtaque de fuerza bruta sobre 'Qklbsov k
   Oczsxyck':)
6 fuerza_bruta_cesar(Qklbsov k Oczsxyck)

```

De lo cual nos arroja 26 claves para poder descifrar, tanto el nombre como la frase.

```

... === Cifrado César Nombre===
Original: Gabriela Espinosa
Cifrado (clave=10): Qklbsov k Oczsxyck
Descifrado: Gabriela Espinosa

Ataque de fuerza bruta sobre 'Qklbsov k Oczsxyck':
Clave 0: Qklbsov k Oczsxyck
Clave 1: Pjkarnuj Nbyrwxbj
Clave 2: Oijzqmti Maxqvwal
Clave 3: Nhiyplsh Lzwpuvzh
Clave 4: Mghxokrg Kyvotuyg
Clave 5: Lfgwnjqf Jxunstxf
Clave 6: Kefvmipe Iwtmsrwe
Clave 7: Jdeulhod Hvsqrvd
Clave 8: Icdtkgnc Gurkpqac
Clave 9: Hbcsjymb Ftqjoptb
Clave 10: Gabriela Espinosa
Clave 11: Fzaqhdzk Drohmnrz
Clave 12: Eyzpgcjy Cqnglmqy
Clave 13: Dxyofbix Bpmfklpx
Clave 14: Cwxneahw Aolejkow
Clave 15: Bvwm dzgv Znkdjinv
Clave 16: Auvlcyfu Ymjchimu
Clave 17: Ztubxet Xlibghlt
Clave 18: Ystjawns Wkhafgks
Clave 19: Xrsizvcr Vjgzeffr
Clave 20: Wqrhyubq Uifydeiq
Clave 21: Vpqgtap Thexcdhp
Clave 22: Uopfwsoz Sgdwbcgo
Clave 23: Tnoevryn Rfcvabfn
Clave 24: Smnduqxm Qebzaem
Clave 25: Rlmctpwl Pdatyzdl

```

Fig. 2. Resultado del ataque por fuerza bruta, Nombre

B. Criptografía Simétrica Moderna (Fernet/AES)

Primero se usa la librería cryptography para cifrar/decifrar un mensaje o archivo de la siguiente manera:

```

1 from Crypto.Cipher import AES
2 from Crypto.Util.Padding import pad, unpad
3 from Crypto.Random import get_random_bytes
4 import binascii
5 def aes_ejemplo():
6     print(\n=== Cifrado AES (Modos ECB y CBC) ===)
7     clave_secreta = get_random_bytes(16)
8     texto_original = No hay lugar como 127.0.0.1
9     datos_bytes = texto_original.encode('utf-8')

```

1) *Comparación con cifrados clásicos:* La evolución desde los cifrados clásicos hacia la criptografía simétrica moderna representa un salto cualitativo en términos de seguridad, fundamentos matemáticos y resistencia al cripto-análisis.

Los cifrados clásicos como César y Vigenère se basan en operaciones aritméticas simples (sustitución y permutación) que operan directamente sobre caracteres individuales [6]. En contraste, la criptografía moderna, ejemplificada por AES, se fundamenta en álgebra abstracta (cuerpos de Galois), teoría de números y redes de sustitución-permutación (SPN) [6]. Esta base matemática compleja permite construir transformaciones no lineales que resisten el cripto-análisis.

2) *Discusión de modos de operación:* Mientras que el cifrado César ofrece apenas 26 claves posibles y Vigenère ofrece

(donde L es la longitud de la clave), estos espacios son trivialmente explorables mediante fuerza bruta o análisis de frecuencia si la clave es corta [6]. AES-256, por su parte, posee un espacio de claves de

$$2^{256}$$

combinaciones, lo que hace impracticable cualquier ataque exhaustivo con la tecnología actual [8].

Los cifrados clásicos dependían del secreto del algoritmo: la seguridad se debilitaba si el método era conocido (violando el principio de Kerckhoff) [9]. En la criptografía moderna, el algoritmo es público y está estandarizado (AES es FIPS 197); toda la seguridad reside exclusivamente en la clave secreta. Esto permite un escrutinio abierto y continuo por parte de la comunidad científica.

Los cifrados clásicos son inherentemente deterministas: el mismo mensaje con la misma clave produce idéntico cifrado, lo que facilita el análisis de patrones. La criptografía moderna introduce modos de operación como CBC (Cipher Block Chaining) y GCM (Galois/Counter Mode) que utilizan vectores de inicialización (IV) aleatorios, garantizando que un mismo mensaje genere salidas diferentes cada vez [11]. En la implementación, la clase Fernet utiliza AES en modo CBC con IV aleatorio, asegurando esta propiedad.

En síntesis, la criptografía simétrica moderna no es simplemente una versión más compleja de los cifrados clásicos, sino un paradigma completamente diferente: se basa en fundamentos matemáticos sólidos, principios de diseño probados (confusión, difusión, modos seguros) y estandarización abierta.

C. Funciones Hash y HMAC

1) *SHA-256*: Se aplica hashlib para obtener el hash de textos y archivos de la siguiente manera:

```
1 def hash_texto(texto):
2     return hashlib.sha256(texto.encode('utf-8')).hexdigest()
3
4
5 def hash_archivo(ruta):
6     sha = hashlib.sha256()
7
8     with open(ruta, 'rb') as f:
9         for bloque in iter(lambda: f.read(4096),
10                             b''):
11             sha.update(bloque)
12     return sha.hexdigest()
```

Ahora definimos tres palabras muy parecidas (Esternocleidomastoideo, Ezternocleidomastoideo y Esternocleydomastoideo), y solo se cambia una letra para probar el hash de textos, se muestra de la siguiente forma:

```
1 print(f'Hash de \"{texto1}\": {hash_texto(texto1)}')
2 print(f'Hash de \"{texto2}\": {hash_texto(texto2)}')
3 print(f'Hash de \"{texto3}\": {hash_texto(texto3)}')
```

Ahora para hacerlo de un archivo, se crea directamente con código el archivo de la siguiente forma:

```
1 def generar_archivo_hash(archivo_entrada,
2                           archivo_salida_hash=None):
3     if archivo_salida_hash is None:
4         archivo_salida_hash = archivo_entrada + '.sha256'
5     hash_val = hash_archivo(archivo_entrada)
6     with open(archivo_salida_hash, 'w') as f:
7         f.write(hash_val)
8     print(f'Hash guardado en {archivo_salida_hash}')
9     return hash_val
```

Luego para los archivos de texto se le agrega el siguiente texto (El profesor nos coloca cinco uuu), se genera el hash de los tres archivos "Nota aclaratoria: el texto del segundo archivo es igual al primero ya que se quiere también hacer la comparativa entre los hash de los archivos, pero el texto del tercer archivo es el siguiente: ("El profesor nos coloca tres uuu"). Luego se leen los archivos .sha256 para comprobar que los hashes coinciden o no, de la siguiente forma:

```
1 with open(archivo1.txt.sha256, 'r') as f:
2     hash_leido = f.read().strip()
3 print(f'Hash guardado de archivo1.txt: {hash_leido}')
4
5 with open(archivo2.txt.sha256, 'r') as f:
6     hash_leido = f.read().strip()
7 print(f'Hash guardado de archivo2.txt: {hash_leido}')
8
9 with open(archivo3.txt.sha256, 'r') as f:
10    hash_leido = f.read().strip()
11 print(f'Hash guardado de archivo3.txt: {hash_leido}')
```

Y el resultado obtenido es el siguiente:

```
=== Hash SHA-256 de Textos ===
Hash de 'Esternocleidomastoideo': c92954e7506842098e6fc15a31d2c79e0784874c42ad75e74710c093f175a647
Hash de 'Ezternocleidomastoideo': 12b2cc05e320890408534afcedb380ed7089e6649892e6f254fa73b976dd2962
Hash de 'Esternocleydomastoideo': 349423b3711d35f1afa4086cac3db83c8c7405f57513c517040672d927e55afc

--- Generación de archivo de hash para archivo1.txt ---
Hash guardado en archivo1.txt.sha256

--- Generación de archivo de hash para archivo2.txt ---
Hash guardado en archivo2.txt.sha256

--- Generación de archivo de hash para archivo3.txt ---
Hash guardado en archivo3.txt.sha256

Hash guardado de archivo1.txt: f48c72581d199a883e3ede74711ddb4f660f8743cb00a6f90b2a687bd7f260
Hash guardado de archivo2.txt: f48c72581d199a883e3ede74711ddb4f660f8743cb00a6f90b2a687bd7f260
Hash guardado de archivo3.txt: 9182b9c1c5b612de8d1278c62a7d07061f66bc644a6b3b668327537257eb619
```

Fig. 3. Resultado del Hash

2) *Integridad*: Se crea una función para comparar hashes de dos archivos y verificar si son idénticos de la siguiente forma:

```
1 def comparar_archivos_hash(ruta1, ruta2):
2     # Calculamos el hash de cada archivo y vemos si son iguales
3     return hash_archivo(ruta1) == hash_archivo(ruta2)
4
5
6
```

```

7 print(f\narchivo1.txt y archivo2.txt iguales?
   {comparar_archivos_hash('archivo1.txt', '
   archivo2.txt')})

8

9 print(farchivo1.txt y archivo3.txt iguales? {
   comparar_archivos_hash('archivo1.txt', '
   archivo3.txt')})

10

11 print(farchivo2.txt y archivo3.txt iguales? {
   comparar_archivos_hash('archivo2.txt', '
   archivo3.txt')})

```

Se hace la comparación entre los tres archivos creados anteriormente, se evidencia que efectivamente, al compararse el primer archivo con el segundo, nos da un valor a "True" lo cual indica que ambos son iguales, y ya comparándolo con el tercer archivo nos arroja un valor de "False" ya que el archivo 3 no es idéntico a los anteriores.

3) **HMAC**: Se programa la generación y verificación de HMAC con SHA-256 y clave de la siguiente manera:

```

1 def generar_hmac(mensaje, clave_hmac):
2     return hmac.new(clave_hmac.encode(), mensaje.
   encode(), hashlib.sha256).hexdigest()
3
4 # Función para verificar si un HMAC recibido es
   correcto
5 def verificar_hmac(mensaje, clave_hmac,
   hmac_recibido):
6     return hmac.compare_digest(calculado,
   hmac_recibido)

```

Para realizar la prueba se usa el mensaje "¿Cómo vendió el horticultor sus productos en la red oscura? Utilizó el enrutamiento de cebolla. la clave secreta fue "Cebollas" y se genera el HMAC para ese mensaje:

```

1 hmac_val = generar_hmac(msg, key_hmac)
2 print(fHMAC calculado: {hmac_val})

```

Ahora se verifica que el HMAC sea el correcto

```

1 print(fVerificación correcta: {verificar_hmac(msg
   , key_hmac, hmac_val)})

```

Y para verificar que el mensaje fue modificado, se le añade una "X" al final, dando como resultado:

```

=== HMAC ===
HMAC calculado: 65d4375abc03cb0eed246fdccfa2d1ca3e25ae9f071ba7ddcfbda564ca98e55
Verificación correcta: True
Verificación con mensaje alterado: False

```

Fig. 4. Resultado del HMAC

D. Autenticación con Hash y Sal

Ahora se simula el registro/inicio de sesión guardando el hash (usa SHA-256), para así integrar el concepto de "sal" y evitar ataques de diccionario, de la siguiente forma:

```

1 base_usuarios = {}
2 def registrar(usuario, password):
3     # Generamos una sal aleatoria de 16 bytes
   (128 bits) y la convertimos a hexadecimal

```

```

4     sal = os.urandom(16).hex()
5     hash_pw = hashlib.pbkdf2_hmac('sha256',
   password.encode(), sal.encode(), 100000).hex
   ()
6     # Guardamos el hash y la sal en nuestra base
   de datos
7     base_usuarios[usuario] = {hash: hash_pw, sal:
   sal}
8     print(fUsuario '{usuario}' registrado con é
   xito.)
9
10 def login(usuario, password):
11     # Si el usuario no existe, devolvemos False
12     if usuario not in base_usuarios:
13         return False
14     # Recuperamos los datos del usuario con el
   hash guardado y sal
15     datos = base_usuarios[usuario]
16     hash_calculado = hashlib.pbkdf2_hmac('sha256',
   password.encode(), datos[sal].encode(),
   100000).hex()
17     return hmac.compare_digest(hash_calculado,
   datos[hash])

```

Ya que se hizo el registro e inicio de seccion, se le implementa un menu interactivo para probar el sistema, con las opciones de 1. Registrar nuevo usuario""2. Iniciar sesión""3. Salir", dando como resultado:

```

--- Menú de Autenticación ---
1. Registrar nuevo usuario
2. Iniciar sesión
3. Salir
Seleccione una opción: 2
Usuario: hola
Contraseña: adios
Usuario o contraseña incorrectos.

--- Menú de Autenticación ---
1. Registrar nuevo usuario
2. Iniciar sesión
3. Salir
Seleccione una opción: 1
Ingrese nombre de usuario: hola
Ingrese contraseña: adios
Usuario 'hola' registrado con éxito.

--- Menú de Autenticación ---
1. Registrar nuevo usuario
2. Iniciar sesión
3. Salir
Seleccione una opción: 2
Usuario: hola
Contraseña: adios
Inicio de sesión exitoso. ¡Bienvenido!

--- Menú de Autenticación ---
1. Registrar nuevo usuario
2. Iniciar sesión
3. Salir
Seleccione una opción: 3
Saliendo del modo interactivo.

```

Fig. 5. Resultado de la autenticación con Hash y sal

E. Firma Digital con RSA

1) *Generación de claves:* Generamos un par de claves RSA de 2048 bits

```
1 key = RSA.generate(2048) # RSA.generate(2048)
   crea un objeto que contiene la clave privada
   y la pública
2 private_key = key.export_key()
3 public_key = key.publickey().export_key()
```

2) *Firma:* Ahora, se define el mensaje que se va a firmar, luego se imprime el mensaje original, se calcula el hash SHA-256 del mensaje y se firma el mensaje.

```
1 firmador = pkcs1_15.new(key)
2 firma = firmador.sign(h)
3 print(fFirma generada (hex): {firma.hex()
   [:64]}...)
4 verificador = pkcs1_15.new(key.publickey())
```

Ya que el hash es como una "huella digital" del mensaje, de tamaño fijo. Ahora se pasa el hash original y la firma, y mostramos que si el mensaje cambia, la verificación falla.

```
1 verificador.verify(h, firma)
2 print(Verificación exitosa: la firma es v
   ávida.)
3 except (ValueError, TypeError):
4     print(Firma inválida.)
5
6 mensaje_modificado = Por qué la IA se negó a
   jugar al escondite? no se.encode('utf-8')
7 print(fMensaje modificado: {
   mensaje_modificado.decode('utf-8')})
8 h_mod = SHA256.new(mensaje_modificado)
9 try:
10     verificador.verify(h_mod, firma)
11     print(La verificación con mensaje
   modificado pasa.)
12 except (ValueError, TypeError):
13     print(Verificación fallida: el mensaje
   fue alterado.)
14
15
16
17 # Llamamos a la función para que se
   ejecute
18 if __name__ == '__main__':
19     firma_digital()
```

Y nos da como resultado que se crean las claves correctamente y se evidencian las verificaciones, tanto que la firma es válida como cuando el mensaje fue alterado.

3) *Verificación:* Ahora se muestra el resultado de la verificación de la firma:

```
--- Firma Digital RSA ---
Claves RSA generadas.
Mensaje original: ¿Por qué la IA se negó a jugar al escondite? ¿Porque sabía que no podía ocultar sus violaciones de datos?
Firma generada (hex): 8e367361869e9e184e7a1a271fd30bca0bed1ff0f8a55060aaf312d422...
Verificación exitosa: la firma es válida.
Mensaje modificado: ¿Por qué la IA se negó a jugar al escondite? no se
Verificación fallida: el mensaje fue alterado.

Claves guardadas en 'clave_privada.pem' y 'clave_publica.pem'.
```

Fig. 6. Verificación de la firma

4) *Fallo al modificar mensaje:* Se ejecuto con el siguiente código:

```
1 mensaje_modificado = Por qué la IA se negó a
   jugar al escondite? no se.encode('utf-8')
2 print(fMensaje modificado: {
   mensaje_modificado.decode('utf-8')})
3 # Calculamos el hash del mensaje modificado
4 h_mod = SHA256.new(mensaje_modificado)
5 try:
6     # Intentamos verificar la misma firma (la
   original) contra el hash del mensaje
   modificado
7     verificador.verify(h_mod, firma)
8     # (no debería pasar porque el mensaje es
   diferente)
9     print(La verificación con mensaje
   modificado pasa.)
10 except (ValueError, TypeError):
11     # la verificación falla porque el mensaje
   fue alterado
12     print(Verificación fallida: el mensaje
   fue alterado.)
```

Y así nos da como resultado que se genera las claves para la firma digital, nos imprime el mensaje original y la firma que se generó (hex), y con esto se realiza una verificación para determinar si el mensaje fue alterado o no, así como se muestra en la imagen.

F. Simulación de Blockchain

1) *Estructura del bloque:* Se implementa una estructura simple de bloque en Python (con campos de datos, timestamp, hash del bloque anterior y propio) de la siguiente manera:

```
1 class BloqueEstudiante:
2     # Bloque que almacena la información de un
   estudiante.
3     def __init__(self, nombre, edad, codigo,
   genero, hash_anterior=):
4         # Guardamos los datos del estudiante
5         self.timestamp = datetime.datetime.now().
   isoformat()
6         self.nombre = nombre
7         self.edad = edad
8         self.codigo = codigo
9         self.genero = genero
10        self.hash_anterior = hash_anterior #
   Hash del bloque anterior en la cadena
11        # Calculamos el hash de este bloque
12        self.hash = self.calcular_hash()
13
14    def calcular_hash(self):
15
16        # Calcula el hash SHA-256 concatenando
   todos los campos del bloque.
17
18        contenido = (str(self.timestamp) + self.
   nombre + str(self.edad) +
19                    self.codigo + self.genero +
   self.hash_anterior)
20        # Aplicamos SHA-256 a ese texto y
   devolvemos el resultado en hexadecimal
21        return hashlib.sha256(contenido.encode())
   .hexdigest()
22
23    def __repr__(self):
24        #Para mostrar el bloque de forma legible.
25        return fBloque(\n
26                f timestamp      : {self.timestamp
   }\n
27                f nombre        : {self.nombre}\n
28                f edad          : {self.edad}\n
```

```

29         f código      : {self.codigo}\n
30         f género      : {self.genero}\n
31         f hash_anterior: {self.
hash_anterior[:10]}\n
32         f hash         : {self.hash
[:10]}\n)

```

2) *Encadenamiento*: El encadenamiento de cada bloque se realiza de la siguiente forma:

```

1 class BlockchainEstudiantil:
2     # Cadena de bloques simple, sin prueba de
trabajo.
3     def __init__(self):
4         # La cadena comienza con un bloque gé
nesis
5         self.cadena = [self.crear_bloque_genesis
()]
6
7     def crear_bloque_genesis(self):
8         # Crea el primer bloque de la cadena (sin
datos reales).
9         return BloqueEstudiante(
10             nombre=Génesis,
11             edad=0,
12             codigo=000000,
13             genero=N/A,
14             hash_anterior=0 # El primer bloque
no tiene anterior
15         )
16
17     # Devuelve el último bloque de la cadena
18     def obtener_ultimo_bloque(self):
19         return self.cadena[-1]
20
21     # Agrega un nuevo estudiante a la cadena (
crea un bloque y lo añade)
22     def agregar_estudiante(self, nombre, edad,
codigo, genero):
23         # Creamos un nuevo bloque; el hash
anterior es el hash del último bloque actual
24         nuevo = BloqueEstudiante(
25             nombre, edad, codigo, genero,
26             hash_anterior=self.
obtener_ultimo_bloque().hash
27         )
28         # Lo añadimos a la cadena
29         self.cadena.append(nuevo)
30         print(fEstudiante registrado: {nombre} (c
ódigo {codigo}))

```

3) *Ruptura de integridad*: Ahora para realizar la ruptura de integridad de la cadena, de la siguiente manera:

```

1 def es_cadena_valida(self):
2
3     # Empezamos desde el bloque 1 (el índice
0 es el génesis)
4     for i in range(1, len(self.cadena)):
5         actual = self.cadena[i]
6         anterior = self.cadena[i-1]
7
8         # Recalculamos el hash del bloque
actual
9         if actual.hash != actual.
calcular_hash():
10             print(fERROR: El hash del bloque
de {actual.nombre} no coincide.)
11             return False
12
13         # Verificamos que el enlace con el
anterior sea correcto

```

```

14         if actual.hash_anterior != anterior.
hash:
15             print(fERROR: El bloque de {
actual.nombre} no apunta correctamente al
anterior.)
16             return False
17
18         # Si todo está bien, la cadena es válida
19         return True
20
21     # Muestra todos los bloques de la cadena
22     def mostrar_cadena(self):
23         # Imprime todos los bloques de la cadena.
24         print(\n=== CADENA DE BLOQUES ESTUDIANTIL
===\n)
25         for i, bloque in enumerate(self.cadena):
26             print(f--- Bloque {i} ---)
27             print(bloque)
28             print()

```

Se hace una verificación de toda la cadena, ya que cada bloque tiene el hash correcto, Cada bloque apunta al hash correcto del bloque anterior, empezando desde el bloque uno hasta el actual, y si en ese re-cálculo se evidencia un mínimo cambio, entonces el sistema arroja un mensaje de error diciendo que efectivamente hubo una ruptura de integridad, así como se muestra en la siguiente imagen:

```

--- Bloque 4 ---
Bloque(
  timestamp      : 2026-02-27T22:10:55.372131
  nombre         : Maria
  edad          : 23
  código         : 1202412
  género         : F
  hash_anterior  : 44c74a2b1a...
  hash           : 5a79ec92a0...
)

Verificando integridad de la cadena...
Cadena válida: True

--- Modificando un bloque (ataque) ---
Bloque original de Gabriela: edad=21
Bloque modificado: ahora edad=99 (Hash sin actualizar)

Verificando cadena después de la modificación...
ERROR: El hash del bloque de Gabriela no coincide.
La cadena ha sido alterada y ya no es válida.

```

Fig. 7. Ruptura de integridad - Blockchain

G. Laboratorio VPN y HTTPS

Ahora se identifica, cómo se implementa el cifrado de canal seguro y sus beneficios mediante un análisis práctico de VPN y HTTPS, combinando observación de herramientas externas con implementación de la siguiente forma:

1) *Cambios observados con VPN*: Se realizó antes de ingresar al link un texto de la ip, y se comprobó el cambio de la IP real a la IP del servidor VPN, ocultando la ubicación geográfica real de nosotros.

2) *Diferencias entre HTTP y HTTPS*: Se implementó un script en colab utilizando las librerías anteriormente mencionadas para demostrar las diferencias entre conexiones HTTP y HTTPS de la siguiente forma:

```
1 def prueba_https():
2     print(=== Laboratorio HTTPS ===\n)
3
4     # 1. Hacemos una petición a un sitio que usa
5     #     HTTPS (conexión segura)
6     url_https = https://www.umng.edu.co/inicio
7     print(f[HTTPS] Solicitando {url_https} ...)
8     try:
9         # Añadimos un User-Agent para que el
10        # servidor no nos bloquee por ser un script
11        req = urllib.request.Request(url_https,
12        headers={'User-Agent': 'Mozilla/5.0'})
13        # Hacemos la petición y esperamos
14        respuesta
15        with urllib.request.urlopen(req, timeout
16        =5) as respuesta:
17            # Mostramos el código de respuesta
18            (200 = OK, 404 = no encontrado, etc.)
19            print(f Código de respuesta: {
20            respuesta.status} {respuesta.reason})
21            print( Encabezados relevantes:)
22            # Recorremos todos los encabezados de
23            # la respuesta
24            for header, value in respuesta.
25            getheaders():
26                # Solo mostramos algunos
27                encabezados
28                if header.lower() in ('content-
29                type', 'server', 'date'):
30                    print(f {header}: {value})
31            # Leemos los primeros 200 bytes del
32            # contenido (para no saturar la pantalla)
33            contenido = respuesta.read(200)
34            print(f Primeros 200 bytes del
35            contenido: {contenido[:100]}...\n)
36        except urllib.error.URLError as e:
37            # Si hay algún error (red, certificado,
38            # etc.), lo mostramos
39            print(f Error en petición HTTPS: {e}\n)
40
41        # 2. Hacemos una petición a un sitio que usa
42        #     HTTP (sin cifrar)
43        url_http = http://httpbin.org/
44        print(f[HTTP] Solicitando {url_http} ...)
45        try:
46            req = urllib.request.Request(url_http,
47            headers={'User-Agent': 'Mozilla/5.0'})
48            with urllib.request.urlopen(req, timeout
49            =5) as respuesta:
50                print(f Código de respuesta: {
51                respuesta.status} {respuesta.reason})
52                print( Encabezados relevantes:)
53                for header, value in respuesta.
54                getheaders():
55                    if header.lower() in ('content-
56                    type', 'server', 'date'):
57                        print(f {header}: {value})
58                # Avisamos de que esta conexión no es
59                # segura
60                print( (Conexión sin cifrar, no es
61                segura)\n)
62            except urllib.error.URLError as e:
63                print(f Error en petición HTTP: {e}\n)
```

Se realizan las peticiones a tres url, siendo uno seguro, no seguro, y otra con certificado expirado, al realizar Conexión tanto al HTTPS de la universidad, como a los no seguros arroja un mensaje de conexión exitosa, de la siguiente manera:



```
--- Laboratorio HTTPS ---
[HTTPS] Solicitando https://www.umng.edu.co/inicio ...
Código de respuesta: 200 OK
Encabezados relevantes:
Content-Type: text/html; charset=UTF-8
Date: Fri, 27 Feb 2020 22:14:26 GMT
Primeros 200 bytes del contenido: <!DOCTYPE html> <html class="ltr" dir="ltr" lang="es-ES"> <head> <title>Universidad Militar Nueva Gr...
[HTTP] Solicitando http://httpbin.org/ ...
Código de respuesta: 200 OK
Encabezados relevantes:
Date: Fri, 27 Feb 2020 22:14:27 GMT
Content-Type: text/html; charset=utf-8
Server: gunicorn/19.9.0
(Conexión sin cifrar, no es segura)
Solicitando https://expired.badssl.com/ ...
Certificado expirado detectado.
Detalle: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: certificate has expired (_ssl.c:1080)
```

Fig. 8. Mensaje Laboratorio VPN y HTTPS

3) *Certificados SSL/TLS*: Aquí se intenta acceder a un site con certificación expirada de la siguiente forma:

```
1 url_expirado = https://expired.badssl.com/
2 print(fSolicitando {url_expirado} ...)
3 try:
4     # Intentamos abrir la URL (por defecto,
5     # Python verifica los certificados)
6     with urllib.request.urlopen(url_expirado,
7     timeout=5) as respuesta:
8         # Si llegamos aquí, el certificado
9         # sería válido
10        print( El certificado es inválido.\n)
11    except urllib.error.URLError as e:
12        # Comprobamos si el error es por
13        # verificación del certificado
14        if isinstance(e.reason, ssl.
15        SSLCertVerificationError):
16            print( certificado expirado detectado
17            .)
18            print(f Detalle: {e.reason}\n)
19        else:
20            print(f Otro error: {e}\n)
```

Este resultado demuestra que Python, al igual que los navegadores web, rechaza conexiones con certificados inválidos, protegiendo al usuario de posibles ataques de intermediario.

V. DISCUSIÓN

1) **Limitaciones de cifrados clásicos**: Los cifrados clásicos, como los esquemas de sustitución y transposición utilizados históricamente (por ejemplo, César o Vigenère), representan hitos fundamentales en la evolución de la criptografía; sin embargo, presentan limitaciones estructurales significativas cuando se analizan bajo modelos de amenaza modernos. Su seguridad depende en gran medida del secreto del algoritmo y de claves con baja complejidad combinatoria, lo que los hace vulnerables a ataques de análisis de frecuencia, fuerza bruta y modelos como Ciphertext-Only Attack (COA) o Known-Plaintext Attack (KPA). Desde una perspectiva matemática, estos sistemas operan sobre transformaciones lineales o patrones repetitivos que no introducen suficiente entropía ni resistencia a la predicción estadística, lo cual compromete la confidencialidad frente a adversarios con capacidades computacionales básicas.

Adicionalmente, los cifrados clásicos carecen de mecanismos formales para garantizar propiedades fundamentales como integridad, autenticidad o no repudio, limitándose únicamente a la ofuscación del mensaje. En entornos digitales contemporáneos, donde los ataques

son automatizados y se ejecutan a gran escala, estos esquemas no ofrecen seguridad computacional sino únicamente seguridad por oscuridad. Esta limitación evidencia el cambio de paradigma hacia la criptografía moderna, basada en fundamentos matemáticos robustos y en supuestos de complejidad computacional bien definidos. Por tanto, el estudio de los cifrados clásicos conserva valor pedagógico e histórico, pero su aplicación práctica en sistemas actuales resulta técnicamente inviable desde una perspectiva de seguridad integral.

- 2) **Ventajas de criptografía moderna:** La criptografía moderna representa un avance sustancial frente a los esquemas clásicos al fundamentarse en principios matemáticos rigurosos y en la teoría de la complejidad computacional. A diferencia de los cifrados históricos, cuya seguridad dependía de patrones ocultos o claves simples, los algoritmos contemporáneos se diseñan bajo supuestos formales de dificultad matemática, como la factorización de enteros grandes o el problema del logaritmo discreto. Esto permite ofrecer seguridad computacionalmente robusta frente a modelos de ataque avanzados como Chosen-Plaintext Attack (CPA) o Chosen-Ciphertext Attack (CCA). Además, la criptografía moderna no se limita al cifrado de datos, sino que integra funciones hash resistentes a colisiones, esquemas de autenticación basados en HMAC y mecanismos de firma digital que garantizan autenticidad y no repudio, ampliando así el alcance de protección más allá de la simple confidencialidad.

Otra ventaja significativa radica en su aplicabilidad en infraestructuras digitales complejas y entornos distribuidos. Protocolos como TLS permiten proteger comunicaciones en tránsito mediante el intercambio seguro de claves, mientras que los sistemas basados en blockchain aprovechan el encadenamiento criptográfico y el consenso distribuido para asegurar la inmutabilidad de los registros. Esta integración multiplica evidencia que la criptografía moderna no es solo una herramienta matemática, sino un componente estructural de la arquitectura de seguridad en redes, sistemas financieros, comercio electrónico y servicios gubernamentales. En consecuencia, su fortaleza no reside únicamente en la robustez de los algoritmos individuales, sino en su capacidad de interoperar dentro de ecosistemas tecnológicos que requieren escalabilidad, resiliencia y verificación confiable en entornos adversariales.

- 3) **Rol de la firma digital en autenticidad y no repudio:** La firma digital constituye uno de los mecanismos más relevantes dentro de la criptografía moderna al extender la protección más allá de la confidencialidad, garantizando propiedades críticas como autenticidad, integridad y no repudio. Basada en criptografía asimétrica y en funciones hash resistentes a colisiones, la firma digital permite vincular de manera única la identidad del emisor

con el contenido de un mensaje. Desde una perspectiva técnica, el proceso implica la generación de un resumen criptográfico del mensaje y su posterior cifrado con la clave privada del firmante; la verificación se realiza utilizando la clave pública correspondiente. Este esquema garantiza que cualquier modificación del mensaje invalide la firma, asegurando integridad, y que solo el titular legítimo de la clave privada pueda haber generado dicha firma, estableciendo autenticidad verificable.

En términos de no repudio, la firma digital introduce una dimensión jurídica y probatoria en los sistemas electrónicos, ya que impide que el emisor niegue posteriormente la autoría del mensaje firmado, siempre que la clave privada haya sido protegida adecuadamente. Esta propiedad resulta esencial en transacciones financieras, contratos electrónicos y sistemas de certificación digital, donde la confianza no puede depender únicamente de la confidencialidad del canal. No obstante, su eficacia depende de una infraestructura de clave pública (PKI) confiable, de mecanismos seguros de almacenamiento de claves y de políticas de certificación robustas. Por tanto, la firma digital no solo representa una herramienta criptográfica, sino un componente estructural en la construcción de confianza digital en entornos distribuidos y legalmente vinculantes.

- 4) **Por qué blockchain depende de hash e inmutabilidad:** La arquitectura de blockchain se fundamenta esencialmente en el uso de funciones hash criptográficas para garantizar la integridad estructural del sistema. Cada bloque contiene un hash que representa matemáticamente su contenido y, a su vez, incorpora el hash del bloque anterior, generando un encadenamiento criptográfico secuencial. Esta propiedad convierte cualquier alteración en una modificación detectable, ya que un cambio mínimo en los datos produce un hash completamente diferente debido al efecto avalancha. De esta manera, la función hash no solo actúa como mecanismo de verificación de integridad, sino como el elemento que asegura la cohesión matemática de la cadena completa. Sin esta dependencia criptográfica, la estructura distribuida carecería de un método confiable para evidenciar manipulaciones en el historial de transacciones.

La inmutabilidad emerge precisamente de esta interdependencia entre bloques y se fortalece mediante mecanismos de consenso distribuido. Alterar un bloque implicaría recalcular su hash y el de todos los bloques posteriores, además de superar el consenso de la mayoría de los nodos de la red. En sistemas como los basados en Proof of Work o Proof of Stake, este proceso requiere un costo computacional o económico considerable, lo que convierte la manipulación en un evento técnicamente complejo y financieramente inviable en redes de gran escala. Por tanto, blockchain no depende únicamente

de la descentralización, sino de la combinación entre funciones hash resistentes a colisiones e inmutabilidad estructural, elementos que transforman la confianza tradicional basada en intermediarios en confianza sustentada en pruebas criptográficas verificables.

- 5) **Diferencia real entre protección de canal (HTTP-S/VPN) y cifrado de datos:** La protección de canal y el cifrado de datos responden a necesidades de seguridad distintas, aunque complementarias, dentro de una arquitectura digital. Protocolos como HTTPS —basado en TLS— y las VPN establecen un canal seguro entre dos puntos mediante cifrado en tránsito, autenticación del servidor y, en muchos casos, intercambio asimétrico de claves para generar claves de sesión simétricas. Este mecanismo protege la información contra interceptación, espionaje o ataques Man-in-the-Middle durante su transmisión por redes públicas. Sin embargo, una vez que los datos llegan a su destino y se descifran para su procesamiento, la protección del canal deja de actuar. Es decir, HTTPS y VPN aseguran la comunicación, pero no necesariamente garantizan la seguridad del contenido cuando este se encuentra almacenado o expuesto dentro del sistema receptor.

En contraste, el cifrado de datos —ya sea en reposo o a nivel de aplicación— protege directamente la información independientemente del medio de transporte. Al cifrar archivos, bases de datos o campos sensibles con algoritmos robustos, se mantiene la confidencialidad incluso si el almacenamiento es comprometido o si un atacante obtiene acceso físico o lógico al sistema. Desde una perspectiva de seguridad integral, la diferencia fundamental radica en el alcance: la protección de canal resguarda el trayecto, mientras que el cifrado de datos protege el activo digital en sí mismo. Por ello, una estrategia de seguridad madura no puede depender exclusivamente de HTTPS o VPN, sino que debe integrar ambas capas dentro de un modelo de defensa en profundidad que contemple amenazas tanto externas como internas.

VI. ESTADO DEL ARTE

A. ¿Cuál es la aplicación concreta presentada?

La aplicación concreta presentada en el documento consiste en la incorporación de la tecnología blockchain como herramienta de apoyo a la Auditoría Forense para la prevención, detección e investigación de fraudes financieros y operaciones de lavado de activos en el entorno de las criptomonedas, particularmente Bitcoin. El estudio no se limita a describir el funcionamiento de la criptomoneda, sino que analiza cómo la estructura técnica del blockchain —basada en registros distribuidos, funciones hash e inmutabilidad de bloques— puede emplearse para rastrear transacciones y detectar patrones sospechosos.

En este sentido, la propuesta se enfoca en utilizar la trazabilidad inherente al blockchain como mecanismo de soporte probatorio y de control, permitiendo fortalecer los procesos de auditoría mediante el análisis técnico de transacciones digitales. Así, la aplicación concreta radica en integrar el estudio estructural y criptográfico del blockchain dentro de programas de auditoría financiera, con el fin de mejorar la transparencia, la supervisión y la mitigación de riesgos en mercados descentralizados.[18]

B. ¿Qué problema o necesidad aborda?

La problemática que aborda el documento se centra en el crecimiento acelerado y global del uso de criptomonedas, especialmente Bitcoin, en un contexto donde su regulación aún es limitada o inexistente en muchos países. Aunque en ciertas jurisdicciones no es reconocido como moneda de curso legal, su adopción continúa en expansión, lo que genera nuevos desafíos para los sistemas financieros tradicionales y para los organismos de control.

Esta situación plantea riesgos asociados a la fuga de capitales, la evasión fiscal y el lavado de activos, debido al carácter descentralizado, transfronterizo y parcialmente anónimo de las transacciones en blockchain. La ausencia de intermediarios financieros tradicionales y la dificultad para identificar plenamente a los usuarios complican la supervisión estatal y el seguimiento de operaciones sospechosas. En consecuencia, surge la necesidad de desarrollar mecanismos técnicos y metodológicos, como la auditoría forense apoyada en el análisis de blockchain, que permitan fortalecer la transparencia, mitigar riesgos financieros y apoyar procesos de control y regulación.

C. ¿Qué desafíos o dificultades se mencionan?

Uno de los principales desafíos que se mencionan en el documento es la comprensión técnica del protocolo criptográfico abierto que sustenta a Bitcoin y al funcionamiento del blockchain. No se trata únicamente de entender la criptomoneda como medio de intercambio, sino de analizar su arquitectura descentralizada, los mecanismos de validación de transacciones, el uso de funciones hash, la minería y el consenso distribuido que garantizan la integridad del sistema. Esta nueva forma de concebir el dinero —digital, descentralizado y sin intermediarios— representa un cambio estructural frente al modelo financiero tradicional, lo que exige una actualización conceptual y técnica por parte de auditores y organismos de control.

Adicionalmente, se señala como dificultad la necesidad de integrar este conocimiento tecnológico dentro de las cuatro fases de la auditoría forense: planeación, ejecución, análisis y comunicación de resultados.[16]. Esto implica adaptar metodologías tradicionales de investigación financiera a un entorno digital complejo, donde las evidencias son criptográficas y las transacciones se registran en redes distribuidas. En consecuencia, el reto no solo es técnico, sino también metodológico

y formativo, pues demanda profesionales con competencias tanto en contabilidad forense como en criptografía y análisis de blockchain.

D. ¿Cuáles son las principales conclusiones o recomendaciones de la publicación?

Las principales conclusiones y recomendaciones de la publicación enfatizan la necesidad de fortalecer el marco institucional y técnico del Estado colombiano frente al crecimiento del uso de criptomonedas. El documento plantea que, aunque la tecnología blockchain ofrece ventajas en términos de innovación financiera, también puede ser utilizada para actividades ilícitas como el lavado de activos, la evasión fiscal y el financiamiento de estructuras criminales. En consecuencia, se recomienda la creación e implementación de controles regulatorios y mecanismos de supervisión que permitan mitigar estos riesgos sin desconocer el avance tecnológico.

Asimismo, se concluye que la Auditoría Forense debe evolucionar para adaptarse a un entorno digital caracterizado por alta volatilidad, complejidad y transformación constante de las modalidades delictivas. Esto implica incorporar herramientas tecnológicas, fortalecer capacidades en análisis de blockchain y capacitar a entidades de control como la Contraloría General de la República (CGR) y la Procuraduría General de la Nación (PGN) en materia de ciberamenazas y delitos informáticos. La recomendación central es avanzar hacia una auditoría forense tecnológicamente preparada, con talento humano especializado y metodologías actualizadas, que permita enfrentar de manera efectiva los desafíos del ecosistema financiero digital.

E. ¿Qué ideas pueden ser útiles para la universidad u organizaciones similares?

Una de las ideas más relevantes que puede resultar útil para la universidad y organizaciones similares es la implementación de políticas y herramientas tecnológicas orientadas a prevenir el fraude documental, especialmente en la emisión y validación de títulos académicos. Casos anteriores han evidenciado la existencia de profesionales que ejercen con documentación falsificada, lo que afecta la reputación institucional y la confianza pública. En este sentido, el uso de mecanismos de auditoría forense y verificación digital permite detectar inconsistencias y validar la autenticidad de certificados académicos.

Adicionalmente, una recomendación estratégica consiste en incorporar tecnología basada en blockchain para la emisión de diplomas, certificaciones y constancias académicas. Al registrar estos documentos en una red blockchain, se garantiza su integridad, inmutabilidad y trazabilidad, permitiendo que cualquier empleador o institución pueda verificar su autenticidad en tiempo real. Esta solución no solo reduce el riesgo de adulteración o falsificación, sino que fortalece la transparencia institucional y moderniza los procesos administrativos. De esta manera, las universidades, instituciones técnicas y colegios

podrían adoptar modelos de certificación digital segura que contribuyan a combatir el fraude profesional y a consolidar la confianza en el sistema educativo.

F. Resumen del análisis del artículo

Título del artículo: [Auditoría forense para mitigar el riesgo de fraude en las inversiones de criptomonedas] Autores: [Corredor Hernandez, Ivonne Marcela] Año y fuente: [2018] – [Respositorio UMNG, Facultad de Ciencias económicas, Revisoría Fiscal] El artículo analiza la relación entre la tecnología blockchain, el uso de criptomonedas especialmente Bitcoin y la Auditoría Forense como herramienta de control frente a riesgos financieros emergentes. El tema central se enfoca en cómo la estructura criptográfica del blockchain, basada en funciones hash, descentralización e inmutabilidad, puede utilizarse no solo como medio de transacción digital, sino como mecanismo técnico de soporte probatorio en investigaciones de fraude y lavado de activos.

Entre los hallazgos clave se identifica que la expansión global de las criptomonedas, incluso en contextos donde no son reconocidas como moneda de curso legal, genera desafíos regulatorios relacionados con evasión fiscal, fuga de capitales y financiamiento de actividades ilícitas. Asimismo, el documento destaca como reto principal la necesidad de comprender la arquitectura técnica del protocolo criptográfico abierto y adaptar las cuatro fases de la auditoría forense a entornos digitales descentralizados. Dentro de las recomendaciones, se propone fortalecer los controles estatales, capacitar a entidades como los organismos de control en ciberamenazas y modernizar los procesos institucionales. Finalmente, se resalta la utilidad de implementar tecnología blockchain en el ámbito universitario para la emisión segura de títulos y certificaciones, garantizando autenticidad, trazabilidad e integridad documental.

Cita textual relevante:

“La auditoría forense debe apoyarse de controles frente a las nuevas formas en que operan los aparatos criminales en un entorno cada vez más volátil, incierto, complejo y ambiguo.”

VII. CONCLUSIONES

- 1) El desarrollo del presente trabajo permitió evidenciar que la criptografía moderna constituye el pilar fundamental de la seguridad en sistemas digitales contemporáneos. La protección efectiva de la información no depende de un único algoritmo, sino de la integración coherente de funciones hash, cifrado simétrico y asimétrico, mecanismos de autenticación y protocolos seguros de comunicación. Esta arquitectura criptográfica permite garantizar propiedades esenciales como confidencialidad, integridad, autenticidad y no repudio.
- 2) Se confirmó que la fortaleza de un sistema criptográfico no radica exclusivamente en la robustez matemática

del algoritmo, sino también en la correcta gestión de claves y en la calidad de la entropía utilizada para su generación. El uso de generadores pseudo-aleatorios no criptográficos puede comprometer completamente la seguridad, mientras que los generadores criptográficamente seguros aseguran imprevisibilidad y resistencia ante ataques de predicción.

- 3) El análisis práctico de la firma digital demostró cómo la combinación de funciones hash y criptografía asimétrica garantiza autenticidad e integridad del mensaje, mientras que el encadenamiento de bloques mediante hash criptográficos evidencia el principio de inmutabilidad en sistemas blockchain. En estos entornos, la seguridad emerge no solo de las propiedades matemáticas de los algoritmos, sino también del consenso distribuido y la descentralización de la validación.
- 4) VPN protege el canal completo, ocultando la IP real y cifra todo el tráfico, siendo especialmente útil en redes no confiables. En cambio HTTPS protege la aplicación, cifrando la comunicación específica con el servidor web, garantizando confidencialidad, integridad y autenticación del sitio.
- 5) Finalmente, se concluye que la seguridad informática debe entenderse como un sistema multi-capa, donde la protección en tránsito (TLS/HTTPS), la protección en reposo y los mecanismos de verificación criptográfica trabajan de manera complementaria para mitigar riesgos en infraestructuras digitales odernas.

REFERENCES

- [1] W. Stallings, *Cryptography and Network Security: Principles and Practice*. Boston: Pearson, 7 ed., 2017.
- [2] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. Boca Raton: CRC Press, 1996.
- [3] National Institute of Standards and Technology, "FIPS PUB 197: Advanced encryption standard (aes)," 2001. Federal Information Processing Standards Publication.
- [4] National Institute of Standards and Technology, "FIPS PUB 180-4: Secure hash standard (shs)," 2015. Federal Information Processing Standards Publication.
- [5] H. Krawczyk, M. Bellare, and R. Canetti, "Hmac: Keyed-hashing for message authentication." RFC 2104, 1997.
- [6] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008. White paper.
- [7] C. E. Shannon, "Communication theory of secrecy systems," *Bell System Technical Journal*, vol. 28, pp. 656–715, Oct. 1949.
- [8] Wikipedia, "Principios de kerckhoffs." https://es.wikipedia.org/wiki/Principios_de_Kerckhoffs, 2023. [Online; accessed 28-Feb-2025].
- [9] VPN Unlimited, "El principio de kerckhoffs." <https://www.vpnunlimited.com/es/help/cybersecurity/kerckhoffs-principle>, 2023. [Online; accessed 28-Feb-2025].
- [10] xsec.sh, "Vulnerabilidades aes-ecb - más allá de ver al pingüino." <https://xsec.sh/blog/vulnerabilidades-ecb/>, 2023. [Online; accessed 28-Feb-2025].
- [11] J. F. Osorio, "Un análisis profundo de los modos de cifrado: Ecb, cbc." Medium, 2023. [Online; accessed 28-Feb-2025].
- [12] Telefónica Tech, "Diferencias entre cifrado, hashing, codificación y ofuscación." <https://telefonicatech.com/blog/diferencias-cifrado-hashing-codificacion-ofuscacion>, June 2022. [Online; accessed 28-Feb-2025].
- [13] CAIB, "2.2.- principios criptográficos." https://sarreplec.caib.es/pluginfile.php/10484/mod_resource/content/2/PSP07_Contenidos_Web/22_principios_criptograficos.html. [Online; accessed 28-Feb-2025].
- [14] M. Bellare and P. Rogaway, "Introduction to modern cryptography." <https://web.cs.ucdavis.edu/~rogaway/classes/227/spring05/book/main.pdf>, 2005. [Online; accessed 28-Feb-2025].
- [15] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. New York, NY, USA: Wiley, 20th anniversary ed., 2015.
- [16] E. Rescorla, "The transport layer security (tls) protocol version 1.3." RFC 8446, 2018. Request for Comments 8446.
- [17] W. Stallings, *Cryptography and Network Security: Principles and Practice*. Boston, MA, USA: Pearson, 8 ed., 2020.
- [18] I. M. Corredor, "Auditoría forense para mitigar el riesgo de fraude en las inversiones de criptomonedas." <https://hdl.handle.net/10654/20070>, s.f. [Online]. Disponible en: <https://hdl.handle.net/10654/20070>.
- [19] C. Paar and J. Pelzl, *Understanding Cryptography: A Textbook for Students and Practitioners*. Berlin, Heidelberg: Springer, 2010.
- [20] D. Kahn, *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet*. New York: Scribner, 1996.
- [21] National Institute of Standards and Technology, "Recommendation for block cipher modes of operation," Tech. Rep. Special Publication 800-38A, NIST, 2001.
- [22] E. Rescorla, "The transport layer security (tls) protocol version 1.3." RFC 8446, 2018.
- [23] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.
- [24] Hornet Security, "Criptografía: Desde los comienzos del cifrado hasta el día de hoy." <https://www.hornetsecurity.com/es/knowledge-base/criptografia/>, 2023. [Online; accessed 28-Feb-2025].
- [25] Whitestack, "¿cómo funciona el cifrado aes? funcionamiento y características." <https://whitestack.com/es/blog/cifrado-aes/>, 2023. [Online; accessed 28-Feb-2025].
- [26] Google Cloud, "¿qué es el encriptado y cómo funciona?." <https://cloud.google.com/learn/what-is-encryption>, 2023.

[Online; accessed 28-Feb-2025].

- [27] Contraloría General de la República, “Herramienta tecnológica apoyada en blockchain para auditoría forense de criptomonedas.” Documento institucional, 2023. Disponible en la biblioteca digital.