DOCUMENTACIÓN DE PROYECTO – ALGORITMOS HASH EN PYTHON

HEILYN YURIMAR GUERRERO AYALA CODIGO: 1004912779

1. Introducción

En la actualidad, los algoritmos de **hash criptográfico** juegan un papel fundamental en la seguridad informática. Son funciones matemáticas que transforman una entrada de datos (texto, archivo, contraseña, etc.) en una **cadena alfanumérica de longitud fija**, conocida como resumen, huella digital o digest. Estas funciones poseen tres características principales:

1. **Determinismo:** el mismo mensaje siempre genera el mismo hash.
2. **No reversibilidad:** no es posible obtener la entrada original a partir del hash.
3. **Sensibilidad a cambios:** una mínima modificación en el mensaje produce un hash completamente diferente.

Los algoritmos hash son ampliamente utilizados en:

- **Verificación de integridad de archivos** (para detectar alteraciones o corrupción de datos).
- **Almacenamiento seguro de contraseñas** (en bases de datos).
- **Autenticación y firmas digitales.**
- **Protocolos de seguridad** como SSL/TLS y sistemas blockchain.

Propósito del proyecto

El presente proyecto tiene como objetivo **implementar y comparar diversos algoritmos hash criptográficos** utilizando el lenguaje de programación **Python**, a fin de analizar su funcionamiento, características y nivel de seguridad.

Los algoritmos abordados son los siguientes:

- **MD5 (Message Digest 5)**
- **SHA-1 (Secure Hash Algorithm 1)**
- **SHA-256 (Secure Hash Algorithm 2)**
- **SHA-3 (Keccak)**
- **HMAC (Hash-based Message Authentication Code)**

Objetivos específicos

- Comprender el funcionamiento teórico de las funciones hash.
- Implementar cada algoritmo en Python utilizando las librerías hashlib y hmac.
- Evaluar las diferencias en la longitud de salida, complejidad y nivel de seguridad.
- Documentar el proceso de desarrollo y resultados obtenidos.

2. Fundamentos Teóricos

2.1 MD5 (Message Digest 5)

El algoritmo **MD5** fue desarrollado por **Ron Rivest** en 1991 como evolución de MD4. Su propósito original fue generar **resúmenes de mensajes de 128 bits** que permitieran verificar la integridad de los datos de forma rápida.

El proceso consiste en dividir el mensaje en bloques de 512 bits, aplicar operaciones lógicas (AND, OR, XOR, rotaciones, sumas modulares) y producir un resultado fijo de 128 bits representado en 32 caracteres hexadecimales.

Características:

- Longitud del hash: 128 bits.
- Librería en Python: hashlib.md5().
- Velocidad: Alta.
- Seguridad: Vulnerable (susceptible a colisiones).

Uso actual: Aunque fue ampliamente empleado en contraseñas, certificados y comprobaciones de integridad, **actualmente se considera inseguro** debido a que pueden encontrarse colisiones de manera práctica. Hoy solo se usa para verificaciones simples o no críticas.

2.2 SHA-1 (Secure Hash Algorithm 1)

El algoritmo **SHA-1** fue diseñado por la **NSA (National Security Agency)** y publicado por el **NIST** en 1995 como parte del estándar FIPS 180-1. Genera un **hash de 160 bits (40 caracteres hexadecimales)**.

Al igual que MD5, procesa bloques de 512 bits, realizando operaciones lógicas y sumas modulares. Su diseño fue una mejora en seguridad sobre MD5, pero con el tiempo también se descubrieron **vulnerabilidades de colisión**.

Características:

- Longitud del hash: 160 bits.
- Librería en Python: hashlib.sha1().
- Seguridad: Débil frente a ataques de colisión.

Uso actual: Obsoleto para aplicaciones criptográficas seguras, pero aún se encuentra en sistemas heredados o para comprobaciones internas (por ejemplo, en Git).

2.3 SHA-2 (SHA-256)

SHA-2 es una familia de funciones hash publicadas en 2001, que incluye variantes como **SHA-224, SHA-256, SHA-384 y SHA-512**. Su diseño mejora significativamente la resistencia frente a colisiones y preimágenes.

El más utilizado es **SHA-256**, que genera un hash de **256 bits (64 caracteres hexadecimales)**, siendo ampliamente empleado en criptomonedas (Bitcoin), certificados digitales, autenticación y almacenamiento seguro de contraseñas.

Características:

- Longitud del hash: 256 bits.
- Librería en Python: hashlib.sha256().
- Seguridad: Alta, sin vulnerabilidades prácticas conocidas.

Uso actual: Es el **estándar moderno** para sistemas que requieren integridad y autenticidad de datos.

2.4 SHA-3 (Keccak)

El algoritmo **SHA-3**, basado en **Keccak**, fue seleccionado en 2012 tras una competencia internacional organizada por el NIST, y se convirtió en estándar oficial en 2015.

A diferencia de SHA-1 y SHA-2, SHA-3 utiliza una estructura llamada **función esponja (sponge function)**, lo que le otorga una arquitectura completamente distinta y mayor resistencia frente a ataques criptográficos.

Características:

- Longitudes disponibles: 224, 256, 384, 512 bits.
- Librería en Python: hashlib.sha3_256(), sha3_512().
- Seguridad: Muy alta.
- Estructura: No basada en Merkle-Damgård, sino en Keccak (sponge).

Uso actual: Recomendado para nuevas aplicaciones que requieran la máxima seguridad, como sistemas de blockchain de nueva generación, autenticación avanzada y firmas post-cuánticas.

2.5 HMAC (Hash-based Message Authentication Code)

El algoritmo **HMAC** no es una función hash por sí misma, sino un **mecanismo criptográfico de autenticación** que utiliza una **clave secreta combinada con una función hash** (por ejemplo, SHA-256 o SHA-512).

Su propósito es garantizar **integridad y autenticidad** del mensaje, protegiendo contra modificaciones y ataques de intermediarios.

El cálculo de un HMAC se realiza mediante la siguiente fórmula:

$$HMAC(K, m) = H((K' \oplus opad) \,||\, H((K' \oplus ipad) \,||\, m))$$

Donde:

- K: clave secreta.
- m: mensaje.
- H: función hash (SHA-256, por ejemplo).
- $opade\ ipad$: constantes de relleno.

Características:

- Depende del hash subyacente (ejemplo: HMAC-SHA256).
- Librería en Python: hmac.new().

- Seguridad: Alta, siempre que la clave sea secreta y segura.

Uso actual: Se emplea en sistemas de autenticación, **APIs seguras**, **JWT (JSON Web Tokens)**, **firmas digitales** y **protocolos HTTPS**.

3.1 Entorno de desarrollo

Para la implementación de los algoritmos hash se utilizó el lenguaje de programación **Python 3.12** debido a su facilidad de uso, soporte de librerías criptográficas y amplia documentación. El proyecto puede ejecutarse en cualquier sistema operativo (Windows, Linux o macOS) que tenga instalado Python y las librerías estándar **hashlib** y **hmac**, las cuales vienen incluidas por defecto.

Requisitos del sistema:

- Python 3.8 o superior
- Editor de texto o IDE (VS Code, PyCharm, IDLE, etc.)
- Conexión a GitHub para almacenar y documentar el código fuente

3.2 Librerías utilizadas

- **Hashlib:** Librería estándar de Python que implementa múltiples algoritmos hash (MD5, SHA-1, SHA-256, SHA-3, entre otros). Permite convertir cadenas de texto en valores hash de manera sencilla.
- **Hmac:** Librería utilizada para generar códigos de autenticación basados en clave secreta (HMAC). Internamente puede emplear algoritmos de la librería hashlib.

3.3. Procedimiento general

1. Se solicita al usuario una entrada de texto o mensaje.
2. El texto se convierte a bytes mediante el método .encode().
3. Se aplica el algoritmo hash seleccionado.
4. Se muestra en pantalla el resultado hexadecimal mediante .hexdigest().

Cada algoritmo fue implementado en un script independiente para facilitar su análisis y comparación.

3.4 Implementación de los algoritmos

a) MD5

```
1   import hashlib
2   texto = input("Ingrese un texto: ")
3   resultado = hashlib.md5(texto.encode())
4   print("MD5:", resultado.hexdigest())
```

Explicación: El texto se transforma en bytes y se pasa como argumento a la función hashlib.md5(). El método .hexdigest() devuelve la huella digital en formato hexadecimal de 32 caracteres.

b) SHA-1

```
6   import hashlib
7   texto = input("Ingrese un texto: ")
8   resultado = hashlib.sha1(texto.encode())
9   print("SHA-1:", resultado.hexdigest())
```

Explicación: Genera un hash de 160 bits (40 caracteres hexadecimales). A pesar de ser similar en uso a MD5, su salida es más larga y con menor riesgo de colisión (aunque hoy es inseguro para uso criptográfico real).

c) SHA-256

```
11  import hashlib
12  texto = input("Ingrese un texto: ")
13  resultado = hashlib.sha256(texto.encode())
14  print("SHA-256:", resultado.hexdigest())
```

Explicación: SHA-256 es uno de los algoritmos más seguros de la familia SHA-2. La salida es de 64 caracteres hexadecimales. Es ampliamente utilizado en firmas digitales y criptomonedas.

d) SHA-3 (Keccak)

```
16  import hashlib
17  texto = input("Ingrese un texto: ")
18  resultado = hashlib.sha3_256(texto.encode())
19  print("SHA-3 (256 bits):", resultado.hexdigest())
```

Explicación: SHA-3 es el sucesor de SHA-2. Emplea una estructura matemática distinta llamada **sponge function**, que mejora la resistencia a ataques. En Python se implementa fácilmente mediante hashlib.sha3_256().

e) HMAC (con SHA-256)

```
21  import hmac
22  import hashlib
23  clave = b"clave_secreta"
24  mensaje = input("Ingrese un mensaje: ").encode()
25  resultado = hmac.new(clave, mensaje, hashlib.sha256)
26  print("HMAC (SHA-256):", resultado.hexdigest())
```

Explicación: Se utiliza la función hmac.new() que requiere tres parámetros:

1. **Clave secreta (key)** en bytes.
2. **Mensaje (message)** en bytes.
3. **Algoritmo hash** (por ejemplo, hashlib.sha256).

El resultado garantiza **integridad y autenticación**, ya que el valor hash solo puede reproducirse si se conoce la misma clave secreta.

3.5 Ejecución

Entrada: Seguridad de la información

Salida: