GENERALITAT VALENCIANA
Conselleria d'Educació, Cultura i Esport

ceedcv CENTRE ESPECÍFIC D'EDUCACIÓ A DISTÀNCIA DE LA COMUNITAT VALENCIANA

Unidad 4.2
Criptografía

Índice

Licencia

1 Hashing y Criptografía simétrica

La seguridad de la información se ha convertido en un pilar fundamental en la era digital, donde la interconexión global y la transferencia constante de datos exigen soluciones robustas para salvaguardar la integridad, confidencialidad y autenticidad de la información.

En esta unidad trabajaremos los fundamentos del *hashing* y la criptografía simétrica, utilizando varios algoritmos fundamentales mediante ejemplos de utilización de estos en el lenguaje de programación Python. Este conocimiento no solo fortalecerá nuestra comprensión de cómo se aseguran los datos sensibles, sino que también nos equipará con habilidades prácticas para aplicar estas técnicas en entornos digitales.

La próxima unidad ampliará nuestra comprensión de la criptografía al introducir la criptografía asimétrica, que incorpora un enfoque de clave pública y privada para una mayor seguridad. Finalmente, exploraremos la criptografía híbrida, fusionando lo mejor de la criptografía simétrica y de la criptografía asimétrica, para crear sistemas de seguridad más completos y eficaces.

- **Hashing**

  El hashing es un proceso matemático mediante el cual se genera una cadena de caracteres única de longitud fija, conocida como "hash" o resumen hash, a partir de datos de entrada.

- **Criptografía Simétrica**

  La criptografía simétrica se basa en el uso de una única llave para cifrar y descifrar la información. La introducción de la llave en este contexto es esencial, ya que es el componente secreto que permite proteger la confidencialidad de los datos. Un ejemplo simple de criptografía simétrica es el Cifrado César, que implica el desplazamiento (la llave) de caracteres en un mensaje.

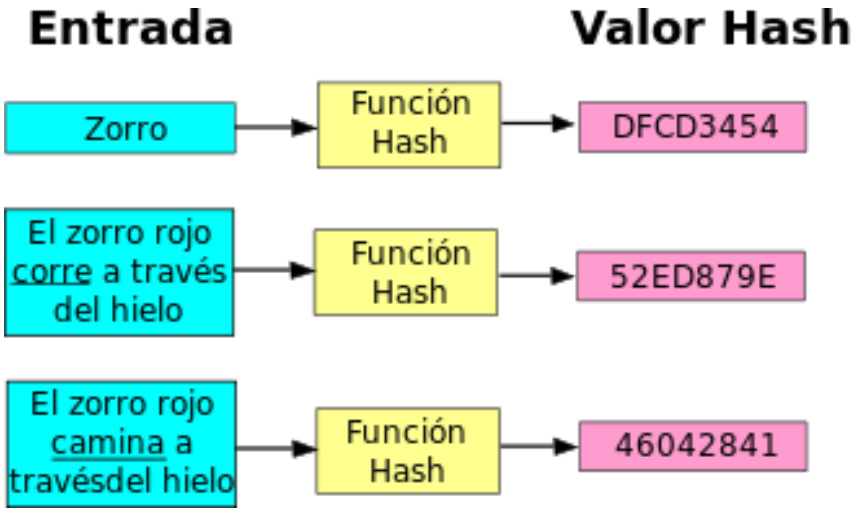
Ambas técnicas se entrelazan para asegurar que los datos mantengan su confidencialidad y su integridad. Con hashing garantizamos la integridad de la información a lo largo del tiempo, mientras que la criptografía simétrica garantiza que la información sensible permanezca confidencial y accesible solo para aquellos que poseen la llave adecuada. La aplicación efectiva de estas técnicas proporciona un marco confiable para salvaguardar la información crítica en las comunicaciones electrónicas y en la gestión de datos sensibles.

2 Hashing

En el ámbito de la seguridad informática, la necesidad de garantizar **la integridad** de los archivos adquiere una relevancia crucial. Cuando los archivos son transmitidos o copiados a través de la red pueden sufrir alteraciones y es por eso que el proceso de hashing se convierte en una herramienta esencial, para garantizar que el archivo en el destino se corresponde con el original. Si el archivo original y el archivo de destino **generan el mismo valor hash**, se puede tener confianza en la integridad del contenido.

El *hashing* utiliza algoritmos para generar un valor hash único o "huella digital" para un conjunto de datos dado. El algoritmo transforma un bloque de datos en una serie de bits con una longitud fija, y por lo general, mucho mas corta que la del bloque analizado.

Este **hash sirve como un identificador único del contenido del archivo**. Cualquier cambio en el archivo, resultará en un valor hash completamente diferente.



Autor: Fercufer, CC BY-SA 3.0, via Wikimedia Commons

Definición:

El hashing es un proceso matemático que toma una entrada (o "mensaje") y produce una cadena de caracteres de longitud fija, que generalmente parece ser aleatoria. Este resultado se conoce como "hash" o "resumen hash".

Características clave:

- La misma entrada siempre dará el mismo hash.
- Pequeños cambios en la entrada deberían generar hashes radicalmente diferentes (propiedad de dispersión).
- Es una función de sentido único: a partir del hash, es extremadamente difícil (teóricamente imposible) obtener la entrada original.

Tipos de alteraciones:

- **Alteraciones fortuitas durante el proceso de manipulación:**

  Cambios accidentales que pueden ocurrir durante la transmisión, copia o manipulación de archivos, lo que subraya la necesidad de mecanismos que verifiquen y mantengan la integridad de los datos.

- **Alteraciones intencionadas con propósitos maliciosos:**

  La posibilidad de manipulaciones deliberadas con la intención de implantar un archivo modificado con fines maliciosos. Este escenario destaca la importancia crítica de implementar medidas de seguridad, como algoritmos hash, para detectar cualquier intento de alteración no autorizada.

Usos comunes:

- **Almacenamiento seguro de contraseñas:**

  En lugar de almacenar contraseñas en texto plano, se almacena el hash de las contraseñas. Esto proporciona una capa adicional de seguridad al proteger las credenciales de los usuarios en caso de que la base de datos sea comprometida.

- **Verificación de la integridad de archivos y mensajes:**

  Los algoritmos de hashing son esenciales para garantizar que los datos, ya sean archivos o mensajes, no hayan sido alterados durante su transmisión o almacenamiento. Comparando los valores hash originales con los actuales, se puede detectar cualquier cambio no autorizado.

- **Creación y verificación de firmas digitales:**

  En el contexto de la seguridad digital, las firmas digitales se generan mediante algoritmos de hashing y criptografía asimétrica. Un individuo puede crear una firma digital al aplicar un algoritmo de hashing al contenido y cifrar ese hash con su clave privada. La verificación implica descifrar la firma con la clave pública del firmante y compararla con un nuevo cálculo del hash del contenido.

- **Búsqueda en bases de datos:**

  Los valores hash se utilizan para optimizar la búsqueda en bases de datos. En lugar de comparar directamente grandes conjuntos de datos, se puede comparar eficientemente solo los valores hash asociados (la "huella digital" o hash es muy corto, por ejemplo 224 bits). Esto acelera el proceso de búsqueda y mejora la eficiencia en la recuperación de información en bases de datos extensas.

- **Uso del hash en la tecnología blockchain:**

  Cada bloque incluye el hash del bloque anterior, formando una cadena inalterable. Cualquier cambio en un bloque modifica su hash, afectando a todos los bloques posteriores y asegurando la detección fácil de manipulaciones de datos.

2.1 Ejemplos de algoritmos de hash

Entre los algoritmos de hash más usados se encuentran el MD5, SHA-1, SHA-2 (que incluye variantes como SHA-224, SHA-256, SHA-384, y SHA-512) y SHA-3 (abarcando SHA3-256, SHA3-384, y SHA3-512). Sin embargo, es crucial señalar que no se recomienda el uso de MD5, SHA-1 ni SHA-224 debido a vulnerabilidades de seguridad bien documentadas.

En particular, un equipo de investigación chino ha demostrado la fragilidad de SHA-1 al romperlo en al menos 2^{69} operaciones, revelando su vulnerabilidad dado que lo han conseguido romper 2000 veces antes que cualquier ataque de fuerza bruta. Esta debilidad resalta la importancia de elegir algoritmos más seguros y resistentes a las amenazas actuales.

Actualmente, el algoritmo más prevalente es SHA-256, preferido por su robustez. Hasta ahora, no ha sido comprometido, y la longitud del código hash resultante, que es de solo 256 bits, ofrece un equilibrio óptimo entre seguridad y eficiencia. SHA-256 proporciona pues una solución confiable y eficiente.

Para más información:

- https://es.wikipedia.org/wiki/Secure_Hash_Algorithm

2.2 Uso en Python de los algoritmos hash

En Python, la librería **hashlib**, integrada en la biblioteca estándar de Python, permite a los desarrolladores implementar funciones de resumen hash de manera eficiente y segura, abarcando diversos algoritmos criptográficos.

En concreto la librería **hashlib** incluye los algoritmos de hash seguro SHA1, SHA224, SHA256, SHA384 y SHA512 (definidos en FIPS 180-2) además del algoritmo MD5 de RSA (definido en Internet RFC 1321).

Las funciones que usaremos de esta librería son:

- **update(data):**

  Actualiza el objeto de hash concatenando los datos anteriores con los nuevos.

- **digest():**

  Retorna el resumen de los datos acumulados hasta este momento en el método update().

- **hexdigest():**

  Como digest() excepto que el resumen es retornado como un objeto de cadena, conteniendo sólo dígitos hexadecimales.

Ejemplo de código en Python usando esta funciones:

```
1  import hashlib
2
3  m = hashlib.sha256()
4  m.update(b"El Libro De Python")
5  salida = m.hexdigest()
6
7  print(salida)
```

Con salida:

```
imple.py"
f7b5c532807800c540f5e4476ea1f6d968294fc34c90f2e7e64435ea3c054ce6
```

Este código ejemplifica el uso de la librería hashlib en Python para calcular el valor hash SHA-256 de una cadena de bytes específica y cómo representar ese hash en formato hexadecimal. El resultado finalmente se imprime por consola.

Explicación detallada:

- **import hashlib**

  Se importa la librería hashlib para acceder a los algoritmos de hashing.

- **m = hashlib.sha256()**

  Se crea un objeto de tipo hash utilizando el algoritmo SHA-256. Este objeto m se utilizará para calcular el valor hash de los datos proporcionados.

- **m.update(b"El Libro De Python")**

  Se actualiza el objeto hash con la cadena de bytes "El Libro De Python". El método update() permite agregar datos al objeto hash. (en formato byte).

- **salida = m.hexdigest()**

  Se calcula el valor hash final almacenado en “update” y se asigna a variable salida. *Nota:* El método hexdigest() devuelve una representación hexadecimal del valor hash.

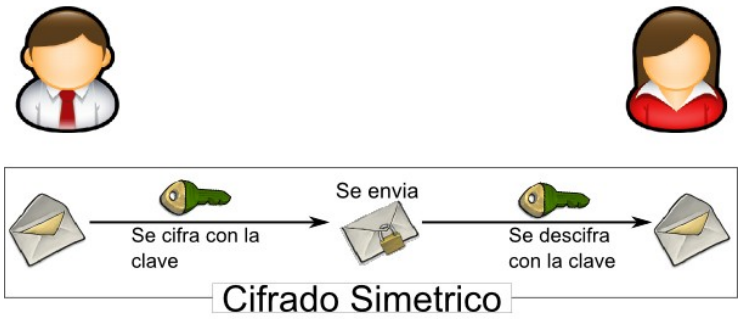
- **print(salida)**

  Se imprime en la consola el valor hash SHA-256 resultante para la cadena de bytes "El Libro De Python".

3 Cifrado simétrico

La criptografía de clave simétrica es un método criptográfico en el cual se utiliza **una única clave** tanto para cifrar como para descifrar la información transmitida entre dos partes. También conocida como **criptografía de clave secreta**, este enfoque implica que el mismo secreto compartido se utiliza en ambas direcciones de la comunicación: desde el remitente al destinatario y viceversa.

En este contexto, las partes comunicantes deben acordar de antemano la clave compartida antes de iniciar la comunicación segura. Una vez que ambas partes tienen acceso a esta clave común, el remitente utiliza dicha clave para cifrar el mensaje antes de transmitirlo, y el destinatario emplea la misma clave para descifrar el mensaje y recuperar la información original.



La criptografía de clave simétrica destaca por su simplicidad y eficiencia en términos de velocidad de procesamiento.

La clave única compartida

La esencia de la seguridad de la criptografía simétrica radica íntegramente en la clave utilizada, mientras que el algoritmo actúa únicamente como un facilitador. La protección efectiva de la seguridad depende, en gran medida, de resguardar la confidencialidad y la complejidad de la clave compartida.

Por ello es esencial adoptar medidas para dificultar cualquier intento de adivinación de la clave, ya que su compromiso podría comprometer la confidencialidad de los mensajes cifrados.

En la actualidad, la velocidad computacional de los ordenadores ha alcanzado niveles sorprendentes, lo que ha aumentado la amenaza potencial contra las claves. Por ello, el tamaño de la clave desempeña un papel fundamental en los criptosistemas modernos. Utilizar claves de mayor longitud contribuye significativamente a la resistencia contra

ataques de fuerza bruta y garantiza una capa adicional de seguridad en un entorno digital cada vez más sofisticado y desafiante.

- **Ventajas de la Criptografía de Clave Simétrica**
  - **Eficiencia y Velocidad**:
    La criptografía de clave simétrica tiende a ser más eficiente y rápida en comparación con su contraparte asimétrica, ya que utiliza la misma clave para cifrar y descifrar.
  - **Menor Sobrecarga Computacional**:
    Al requerir menos recursos computacionales, la criptografía de clave simétrica es más adecuada para aplicaciones donde la eficiencia y la velocidad son prioritarias.
  - **Implementación Sencilla**:
    La implementación de algoritmos de clave simétrica suele ser más sencilla en términos de configuración y gestión en comparación con sistemas asimétricos.
  - **Manejo de Grandes Volúmenes de Datos**:
    Es particularmente eficaz en situaciones que involucran grandes volúmenes de datos y comunicación frecuente.
- **Desventajas de la Criptografía de Clave Simétrica:**
  - **Distribución Segura de Claves:**
    La principal desventaja es la necesidad de distribuir las claves de forma segura entre las partes involucradas, lo cual puede ser un desafío en entornos distribuidos.
  - **Complejidad en Sistemas Multipartes:**
    En sistemas con múltiples partes que requieren comunicarse de manera segura, la gestión y distribución de las claves puede volverse más compleja.
  - **Escalabilidad Limitada:**
    Puede tener limitaciones en términos de escalabilidad en comparación con la criptografía asimétrica, especialmente en escenarios que involucran un gran número de usuarios o dispositivos.
  - **Vulnerabilidad a Ataques de Fuerza Bruta:**
    Claves más cortas pueden ser vulnerables a ataques de fuerza bruta, donde un atacante intenta descifrar el mensaje probando diferentes combinaciones de

claves. Esto subraya la importancia de utilizar **claves lo suficientemente largas** para resistir este tipo de ataques.

3.1 Cifrado por sustitución

El cifrado por sustitución es una técnica criptográfica clásica en la que cada letra o grupo de letras en el texto original es reemplazado por otra letra o grupo de letras de acuerdo con una regla específica. **Esta regla de reemplazo constituye la clave del cifrado**.

Hay varios tipos de cifrado por sustitución:

- **Monoalfabético**:

  En el cifrado monoalfabético, cada letra del alfabeto se sustituye por una única letra específica en el proceso de cifrado. Esto implica que una misma letra siempre se cifrará de la misma manera en todo el mensaje. El más conocido es el cifrado de César.

- **Homofónico**:

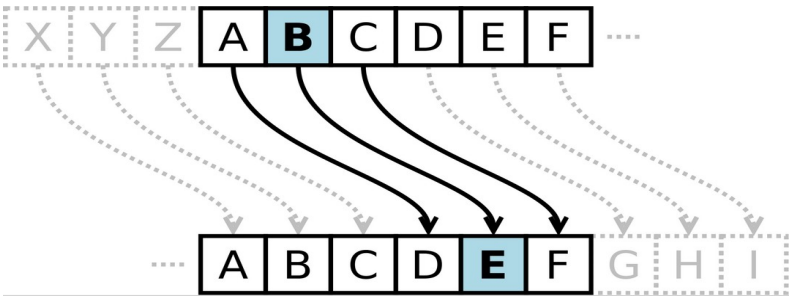
  En el cifrado homofónico, una unidad del texto plano (como una letra) puede ser sustituida por una de entre varias posibilidades existentes. Esto aumenta la complejidad del cifrado al ofrecer múltiples opciones de sustitución para una misma letra.

- **Polialfabético**:

  El cifrado polialfabético utiliza múltiples juegos de sustituciones a lo largo del mensaje, proporcionando variabilidad en el cifrado. Este enfoque hace que la tarea de descifrar sea más compleja, ya que una misma letra puede tener diferentes cifrados según su posición en el mensaje o el juego de sustituciones utilizado.

Cifrado de César:

En el cifrado de César, cada letra en el mensaje original se desplaza hacia adelante o hacia atrás en el alfabeto por un número fijo de posiciones. Por ejemplo, con un desplazamiento de 3, la letra "A" se convertiría en "D", "B" en "E", y así sucesivamente. Este tipo de cifrado es un ejemplo de cifrado de sustitución monoalfabético, ya que cada letra es reemplazada por otra letra del mismo alfabeto.



Cepheus, Public domain, via Wikimedia Commons

3.2 Explorando los Fundamentos de la Criptografía Simétrica

La criptografía simétrica desempeña un papel fundamental en la protección de la información, y su aplicación se extiende a través de dos enfoques principales: cifrado en flujo y cifrado en bloque.

Cifrado en Flujo:

En este método, el cifrado se lleva a cabo bit a bit o byte a byte. Un flujo de claves se genera a partir de la clave inicial, evolucionando a medida que el algoritmo procesa el mensaje original. Esta técnica proporciona una forma dinámica de cifrar la información, adaptándose a la naturaleza del flujo de datos. Entre los cifrados de flujo notables se encuentran RC4 y RC6, que ilustran el poder y la flexibilidad de este enfoque.

Cifrado en Bloque:

En contraste, el cifrado en bloque divide el mensaje en bloques más grandes, comúnmente de 64 o 128 bits. En este caso, la misma clave se utiliza para cifrar cada bloque, y la disposición de la clave determina el orden de sustitución. Algunos de los algoritmos de cifrado simétrico de bloque más destacados incluyen:

- **Data Encryption Standard (DES)**: Aunque ahora considerado obsoleto debido a la longitud de la clave relativamente corta, DES fue pionero en la criptografía de bloque.
- **Triple DES (3DES)**: Un avance sobre DES, 3DES utiliza tres claves consecutivas para fortalecer la seguridad.
- **Advanced Encryption Standard (AES)**: Elegido como el estándar de cifrado por muchos gobiernos y organizaciones, AES ha demostrado ser robusto y seguro en diversas aplicaciones.

3.3 Algoritmo DES

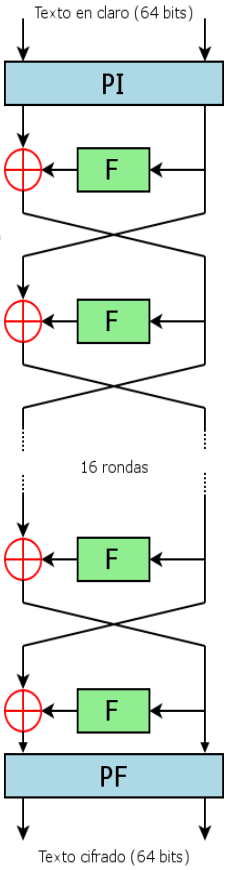
El Data Encryption Standard (DES) es un algoritmo de cifrado simétrico adoptado como estándar federal por el Instituto Nacional de Normas y Tecnología (NIST) de los Estados Unidos en 1977. Aunque DES ha sido reemplazado en muchos casos por algoritmos más avanzados debido a su longitud de clave corta, sigue siendo relevante en el ámbito histórico y educativo.

DES es un algoritmo de cifrado por bloques con un **tamaño fijo de bloque de 64 bits** y utiliza una clave criptográfica de 56 bits para modificar la transformación. El descifrado solo puede ser realizado por aquellos que conocen la clave concreta utilizada en el cifrado. Sin embargo, hoy en día, DES se considera inseguro para muchas aplicaciones, principalmente debido a su corta longitud de clave, que ha permitido romper claves en menos de 24 horas.

Para abordar esta vulnerabilidad, se introdujo Triple DES (TDES o 3DES), que duplica la longitud de la clave a 112 bits, haciéndolo más seguro, aunque también más lento. A pesar de estas mejoras, tanto DES como TDES están siendo reemplazados progresivamente por el algoritmo Advanced Encryption Standard (AES) debido a su mayor seguridad y eficiencia en aplicaciones modernas.

Los pasos principales del algoritmo DES son:

1) Fraccionamiento del mensaje en bloques de 64 bits.
2) Permutación de los bloques.
3) Partición de cada bloque en parte derecha e izquierda
4) Fases de permutación y sustitución repetidas 16 veces, denominadas “rondas”.
5) Reconexión de las dos partes, derecha e izquierda.
6) Permutación inicial inversa.



3.4 Uso en Python del algoritmo DES

El algoritmo Data Encryption Standard (DES) es un cifrado simétrico que puede ser implementado en Python con la ayuda de la biblioteca pycryptodome. A continuación, se presenta un ejemplo de cómo utilizar DES para cifrar y descifrar mensajes, además de utilizar base32hex para la codificación.

Instalación de librerías:

Antes de comenzar, asegúrate de tener instaladas las librerías pycryptodome y base32hex. Puedes instalarlas utilizando pip:

- pip install pycryptodome
- pip install base32hex

El algoritmo DES se usaba sobre todo para encriptar passwords, es por ello que se presenta un ejemplo de código que refleja el proceso básico de cifrado y descifrado utilizando DES en modo OFB. Ten en cuenta que este código es educativo dado que el uso de DES se considera obsoleto en la actualidad debido a su corta longitud de clave y vulnerabilidad.

Las librerías que hemos usado son:

- **Crypto.Cipher** con DES:

  Crypto.Cipher es un módulo de la biblioteca pycryptodome que proporciona implementaciones de varios algoritmos de cifrado.

  - **DES** es una clase dentro de este módulo que implementa el algoritmo de cifrado DES (Data Encryption Standard). DES es una cipher (algoritmo de cifrado) de 8 bytes y por ello los datos deben tener una longitud múltiplo de 8: **Longitud%8=0**. Si tenemos un texto que no cumple esto, **debemos modificarlo añadiendo caracteres**, por ejemplo, espacios al final.
  - **Key** se genera una clave aleatoria para la criptografía simétrica. En este código cada vez se genera una nueva key. Se podría guardar en fichero para evitar esta generación continua de claves nuevas.
  - **IV** es un vector de Vector de Inicialización (Initialization Vector) es un valor único y aleatorio (*os.urandom*) que se utiliza junto con la clave para cifrar datos. El IV se utiliza para introducir aleatoriedad en el proceso de cifrado, lo que es esencial para garantizar que la misma entrada cifrada con la misma clave no produzca siempre la misma salida cifrada. **DES.block_size**: Es una constante que representa el tamaño del bloque del algoritmo DES. Para DES, el tamaño del bloque es de **8 bytes (64 bits)**. Por lo tanto, DES.block_size proporciona el tamaño del bloque que se espera para el IV.

  - **OFB** hay varios modos de cifrado: ECB, CBC, CFB, OFB, CTR, CCM, EAX, GCM … El modo OFB (output feedback) emplea una clave para crear un bloque pseudoaleatorio que es operado a través de XOR con el texto claro para generar el texto cifrado. Requiere de un vector de inicialización (**IV**) que debe ser único para cada ejecución realizada. Para más información:

    https://pycryptodome.readthedocs.io/en/latest/src/cipher/classic.html#ofb-mode

- **base64**:

  base64 es un módulo estándar de Python que proporciona funciones para codificar y decodificar datos en formato base64.

  En este código, base64.b64encode(mensajeCifrado).decode("utf-8") se utiliza para convertir los bytes cifrados en una representación base64 más legible en formato de cadena de texto.

Para crear un nuevo objeto de cifrado se ejecuta: **DES.new(key,DES.MODE_OFB, IV=iv)**:

- La calve key (bytes): Clave criptográfica simétrica
- Modo OFB
- IV: el vector de inicialización debe ser único para la combinación mensaje/texto. Debe tener el tamaño del bloque. Si no se indica la librería crea uno de forma random.

Una vez instanciado el objeto se le aplican las dos operaciones básicas:

- encrypt()
- decrypt()

Cabe destacar que no se puede re-usar un objeto DES para encriptar otro texto. Para encriptar otros textos hay que crear objetos de cifrado diferentes.

```python
from Crypto.Cipher import DES
import base64
import os


def formatear_multiplo_de_8(text):
    while(len(text)%8 != 0):
        text += b' ' #leañado espacios en blanco al final
    return text


def encriptar(mensaje,key,iv):
    #instanciamos un nuevo objeto DES
    cipher = DES.new(key,DES.MODE_OFB,IV=iv)
    #ciframos los datos
    bytesCifrados =  cipher.encrypt(mensaje)
    print ("Bytes cifrados: ", bytesCifrados)

    return bytesCifrados

def desencriptar(mensaje,key,iv):
    #es necesario un nuevo objeto para descifrar
    cipher = DES.new(key, DES.MODE_OFB,IV=iv)
    mensajeDescifrado = cipher.decrypt(mensaje)

    return mensajeDescifrado

def main():
    Password = b'Mi pa$$ para P$P' #Los algoritmos DES se usaban para encrip
    mensaje = formatear_multiplo_de_8(Password)
    print ("Mensaje original:", mensaje.decode())
    key = os.urandom(8) #establecemos una clave de 8 bytes
    iv = os.urandom(DES.block_size) #generamos aleatoriamente un iv del tama

    mensajeCifrado = encriptar(mensaje,key,iv)
    #para imprimir una mejor representación
    mensajeCifrado_decod = base64.b64encode(mensajeCifrado).decode("utf-8")
    print ("Mensaje Cifrado:", mensajeCifrado_decod)

    #desciframos usando la misma key e iv
    mensajeDescifrado = desencriptar(mensajeCifrado,key,iv)
    print ("Mensaje: ", mensajeDescifrado.decode("utf-8"))

if __name__=="__main__":
    main()
```

3.5 Algoritmo AES

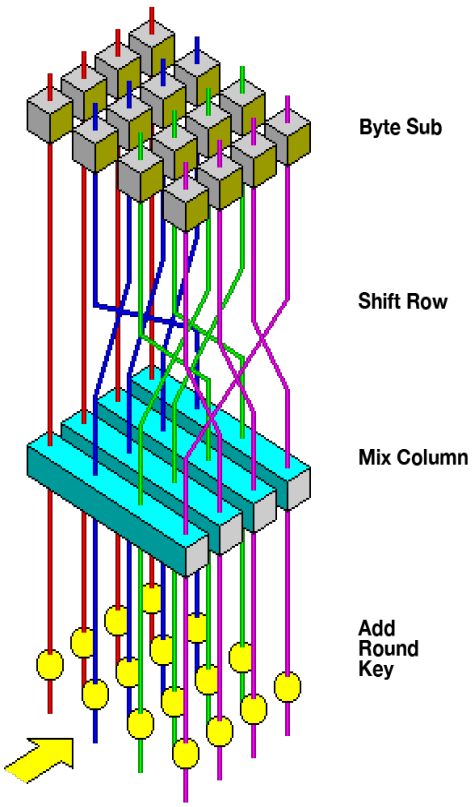
AES (Advanced Encryption Standard), también conocido como Rijndael, es un algoritmo de cifrado simétrico, lo que significa que utiliza la misma clave tanto para cifrar como para descifrar la información. Es un algoritmo de cifrado de bloque, lo que implica que divide los datos en bloques fijos antes de realizar las operaciones de cifrado o descifrado. Cada bloque es tratado de forma independiente.

El **tamaño de bloque es fijo, de 128 bits** y ofrece tres longitudes de clave diferentes: AES-128, AES-192 y AES-256, que utilizan **claves de 128, 192 y 256 bits** respectivamente. Aunque todas las variantes son consideradas seguras, se tiende a preferir AES-256 en situaciones donde se necesita el máximo nivel de seguridad debido a su longitud de clave más larga.

El algoritmo se basa en establecer una matriz de 4 x 4 bytes llamada state, con los 128 bits de los que consta el bloque.

Algunos de los pasos del algoritmo:

- En la ronda inicial se hace una operación XOR de cada byte con el correspondiente de la clave.
- A partir de aquí se hace un proceso llamado SubBytes, que deriva muchas claves a partir de la inicial para las rondas posteriores.
- Se aplica un proceso conocido como RotWord, el cual rota el primer byte hacia el último lugar de la columna, consiguiendo así la primera columna de la siguiente clave.
- La transformación S-Box se hace tomando los primeros cuatro bits como índice de fila y los cuatro siguientes como índice de columna.
- Se hace una XOR de la columna con una constante RCON, diferente para cada ronda.
- Al resto de columnas se le hace una XOR de la columna anterior con la columna de la clave de la ronda previa.
- A partir de ahí se generan rondas intermedias, dependiendo sus repeticiones del tamaño de la clave.



Historia:

La necesidad de un nuevo estándar de cifrado surgió a fines de la década de 1990 cuando el algoritmo DES (Data Encryption Standard) comenzó a mostrar signos de debilidad frente a los ataques de fuerza bruta. En 2001, el NIST (National Institute of Standards and Technology) de los Estados Unidos seleccionó el algoritmo Rijndael, desarrollado por los criptógrafos belgas Vincent Rijmen e Joan Daemen, como el estándar de cifrado para reemplazar al DES. Rijndael fue elegido debido a su seguridad robusta y eficiencia en una variedad de plataformas.

Después de un proceso de estandarización, que duró 5 años, se transformó en un estándar efectivo el 26 de mayo de 2002.

Desde 2006, el AES es uno de los algoritmos más populares usados en criptografía simétrica.

Uso Actual:

AES, en particular el estándar AES-256, se ha consolidado como el algoritmo de cifrado más ampliamente utilizado y confiable en el mundo de la seguridad informática. Su aplicación se extiende a diversas áreas, destacando su presencia en:

- **Encriptación de Ficheros:**

  Utilizado en sistemas como BitLocker de Windows o en la opción de código abierto VeraCrypt para la protección de discos y archivos sensibles.

- **Cifrado de Archivos Comprimidos:**

  Presente en herramientas populares como WinRAR y 7Zip, así como en documentos de las últimas versiones de Microsoft Word, asegurando la confidencialidad de datos incluso en archivos comprimidos.

- **Protocolos de Comunicación Segura:**

  Implementado en plataformas de mensajería cifrada de extremo a extremo como WhatsApp, Telegram y Signal. Esta aplicación garantiza que las conversaciones y archivos compartidos a través de estas plataformas permanezcan confidenciales y seguros.

4 Referencias

https://es.wikipedia.org/wiki/Funci%C3%B3n_hash

https://www.redeszone.net/tutoriales/seguridad/criptografia-algoritmos-hash/

https://quike.it/es/importancia-del-hash-en-la-integridad-de-los-archivos-informaticos/

https://quike.it/es/importancia-del-hash-en-la-integridad-de-los-archivos-informaticos/

https://es.wikipedia.org/wiki/Cifrado_por_sustituci%C3%B3n

https://es.wikipedia.org/wiki/Cifrado_C%C3%A9sar

https://developer.mozilla.org/es/docs/Glossary/Symmetric-key_cryptography

https://es.wikipedia.org/wiki/Data_Encryption_Standard

https://es.wikipedia.org/wiki/Advanced_Encryption_Standard

https://commons.wikimedia.org/wiki/File:AES_(Rijndael)_Round_Function.png

(https://commons.wikimedia.org/wiki/File:AES_(Rijndael)_Round_Function.png)