



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

Práctica 9: Datos personales sensibles y AES

ALUMNOS

Gabriela López Diego - 318243485
Abraham Jiménez Reyes - 318230577
Javier Alejandro Rivera Zavala - 311288876
Juan Daniel San Martín Macias - 318181637

PROFESORA

Anayanzi Delia Martínez Hernández

AYUDANTES

Cecilia del Carmen Villatoro Ramos
Roberto Adrián Bonilla Ruíz
Ivan Daniel Galindo Perez
Roberto Adrián Bonilla Ruíz

ASIGNATURA

Criptografía y Seguridad

09 de Mayo del 2024

1. Introducción

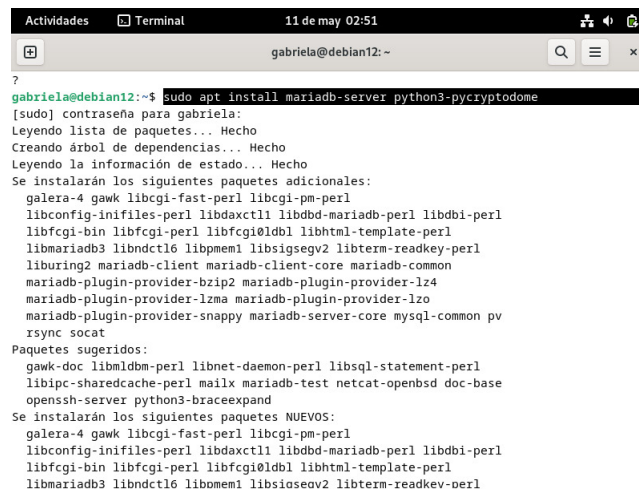
En esta práctica se pretende explorar una simulación de cifrado para datos personales, basado en las leyes que tenemos sobre la de protección de datos personales, en México. Dichas leyes de protección de datos, exigen que se aplique el principio de responsabilidad para garantizar la seguridad y confidencialidad de la información personal. En este sentido, se requiere implementar medidas adecuadas para almacenar de manera segura los datos personales sensibles de los usuarios de un sistema.

En esta práctica, nos propusimos utilizar técnicas de cifrado para proteger la información almacenada en una base de datos relacional. Específicamente, se ha desarrollado un programa que permite almacenar la información de expedientes médicos de los usuarios. Para asegurar la confidencialidad, tanto el diagnóstico como el tratamiento médico se cifran antes de ser almacenados, mientras que el nombre del paciente se guarda en texto claro. El programa implementa una serie de tareas para garantizar la seguridad de los datos. En primer lugar, genera una llave de 256 bits utilizando el algoritmo de derivación de llaves PBKDF2. Además, se generan nonces para el modo de operación CTR, que se utilizan tanto para el cifrado del diagnóstico como del tratamiento médico. Los datos personales son cifrados utilizando el algoritmo AES-256-CTR, y tanto la información cifrada como los nonces empleados se codifican en base64.

2. Desarrollo

2.1. Requisitos

- Sistema Operativo GNU/Linux(Debian 12)
- Manejador de base de datos (Maria DB)
- Python versión 3 con las siguientes paqueterías
 1. PyCryptodome
 2. MySQLdb
 3. SecureString



```
Actividades Terminal 11 de may 02:51
gabriela@debian12: ~
?
gabriela@debian12:~$ sudo apt install mariadb-server python3-pycryptodome
[sudo] contraseña para gabriela:
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes adicionales:
galera-4 gawk libcgi-fast-perl libcgi-pm-perl
libconfig-inifiles-perl libdaxctl1 libdbd-mariadb-perl libdbi-perl
libfcgi-bin libfcgi-perl libfcgi0ldbl libhtml-template-perl
libmariadb3 libndctl6 libmem1 libsigsegv2 libterm-readkey-perl
liburing2 mariadb-client mariadb-client-core mariadb-common
mariadb-plugin-provider-bzip2 mariadb-plugin-provider-lz4
mariadb-plugin-provider-lzma mariadb-plugin-provider-lzo
mariadb-plugin-provider-snappy mariadb-server-core mysql-common pv
rsync socat
Paquetes sugeridos:
gawk-doc libmldbm-perl libnet-daemon-perl libsql-statement-perl
libipc-sharedcache-perl mailx mariadb-test netcat-openbsd doc-base
openssh-server python3-braceexpand
Se instalarán los siguientes paquetes NUEVOS:
galera-4 gawk libcgi-fast-perl libcgi-pm-perl
libconfig-inifiles-perl libdaxctl1 libdbd-mariadb-perl libdbi-perl
libfcgi-bin libfcgi-perl libfcgi0ldbl libhtml-template-perl
libmariadb3 libndctl6 libmem1 libsigsegv2 libterm-readkey-perl
```

Figura 1: Instalación de la paquetería pyCryptodome en la maquina virtual con debian12

```
Actividades Terminal 11 de may 02:52
gabriela@debian12:~
gabriela@debian12:~$ sudo apt install python3-mysqldb
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes NUEVOS:
  python3-mysqldb
0 actualizados, 1 nuevos se instalarán, 0 para eliminar y 0 no actualizados.
Se necesita descargar 49.5 kB de archivos.
Se utilizarán 176 kB de espacio de disco adicional después de esta operación.
Des:1 http://deb.debian.org/debian bookworm/main amd64 python3-mysqldb amd64 1.4.6-2+b1
[49.5 kB]
Descargados 49.5 kB en 0s (192 kB/s)
Seleccionando el paquete python3-mysqldb previamente no seleccionado.
(Leyendo la base de datos ... 155960 ficheros o directorios instalados actualmente.)
Preparando para desempaquetar .../python3-mysqldb_1.4.6-2+b1_amd64.deb ...
Desempaquetando python3-mysqldb (1.4.6-2+b1) ...
Configurando python3-mysqldb (1.4.6-2+b1) ...
gabriela@debian12:~$ sudo apt install python3-securestring
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes NUEVOS:
  python3-securestring
0 actualizados, 1 nuevos se instalarán, 0 para eliminar y 0 no actualizados.
Se necesita descargar 4.680 B de archivos.
```

Figura 2: Instalación de la paquetería mysqldb en la maquina virtual con debian12

```
Actividades Terminal 11 de may 02:52
gabriela@debian12:~
Seleccionando el paquete python3-mysqldb previamente no seleccionado.
(Leyendo la base de datos ... 155960 ficheros o directorios instalados actualmente.)
Preparando para desempaquetar .../python3-mysqldb_1.4.6-2+b1_amd64.deb ...
Desempaquetando python3-mysqldb (1.4.6-2+b1) ...
Configurando python3-mysqldb (1.4.6-2+b1) ...
gabriela@debian12:~$ sudo apt install python3-securestring
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes NUEVOS:
  python3-securestring
0 actualizados, 1 nuevos se instalarán, 0 para eliminar y 0 no actualizados.
Se necesita descargar 4.680 B de archivos.
Se utilizarán 29.7 kB de espacio de disco adicional después de esta operación.
Des:1 http://deb.debian.org/debian bookworm/main amd64 python3-securestring amd64 0.2-2
+b1 [4.680 B]
Descargados 4.680 B en 0s (33.1 kB/s)
Seleccionando el paquete python3-securestring:amd64 previamente no seleccionado.
(Leyendo la base de datos ... 155989 ficheros o directorios instalados actualmente.)
Preparando para desempaquetar .../python3-securestring_0.2-2+b1_amd64.deb ...
Desempaquetando python3-securestring:amd64 (0.2-2+b1) ...
Configurando python3-securestring:amd64 (0.2-2+b1) ...
gabriela@debian12:~$ sudo apt install mariadb-server
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
```

Figura 3: Instalación de la paquetería securestring en la maquina virtual con debian12

Para instalar maria-db en Debian 12, requerimos de hacer uso de los comandos **sudo apt update** para actualizar todos los paquetes del sistema y posteriormente de **sudo apt install mariadb-server**, que instalará en nuestro equipo el sistema de gestión de bases de datos relacionales, Maria DB. Una vez hecho lo anterior, arrancamos el sistema con el comando **sudo systemctl enable --now mariadb**.

2.2. Creación de una base de datos en mariadb para el cifrado AES CTR

Para el cifrado y descifrado AES en modo CTR, utilizaremos una misma base de datos. A dicha base la llamaremos **hospital** y en ella crearemos una única tabla **expediente**. Nuestro esquema estará construido de la siguiente manera

```
1 CREATE DATABASE IF NOT EXISTS hospital;
2 USE hospital;
3
4 CREATE TABLE IF NOT EXISTS expediente (
5     id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
6     nombre varchar(255) NOT NULL,
7     diagnostico varchar(450) NOT NULL,
8     tratamiento varchar(450) NOT NULL,
9     passwordSalt varchar(25) NOT NULL,
10    diag_nonce varchar(12) NOT NULL,
```

```

11 treat_nonce varchar(12) NOT NULL
12 );

```

El archivo sql que contiene los anteriores comandos, se llama *hospital_scheme.sql* y lo encontraremos dentro de la carpeta *cifradoAES.CTR*, para vaciarlo dentro del sistema se hizo uso del comando `sudo mariadb < hospital_scheme.sql`.

Para arrancar el sistema Maria DB, necesitamos acceder como usuario root al sistema a través del comando **sudo mariadb --user root** mismo que nos puede pedir una contraseña, que por defecto es la cadena vacía. Luego de haber hecho lo anterior, ejecutamos los siguientes comandos en Maria DB para crear un nuevo usuario con contraseña y le otorgaremos todos los permisos sobre la base **hospital**.

```

1 CREATE USER 'nuevousuario'@'localhost' IDENTIFIED BY 'debian12';
2
3 GRANT ALL PRIVILEGES ON hospital.* TO 'nuevousuario'@'localhost';
4
5 FLUSH PRIVILEGES;

```

Por lo cual, las credenciales de acceso que utilizarán mas adelante los archivos *cifradoAES.CTR.py* y *descifradoAES.CTR.py* para el cifrado y descifrado, serán las siguientes y las cuales se ubicaran dentro del archivo *config.py*

```

1 # Credenciales de la base de datos MariaDB
2 user = "nuevousuario"
3 password = "debian12"
4 dbname = "hospital"
5 host = "localhost"
6 port = 3306

```

Este archivo también lo encontramos dentro de la carpeta *cifradoAES.CTR*

2.3. Creación del script para el cifrado AES en modo CTR

Dado que ya se nos entrega el código para hacer un cifrado de tipo AES en modo CTR, cuyo funcionamiento ya se encuentra descrito en la descripción de la práctica, lo único que hizo falta fue implementar en primer lugar, una parte del script que tome los datos del archivo *diagnosticos-tratamientos.txt* y los guarde en un diccionario de python, esto dada la estructura con la que están guardados en el archivo de texto plano, que ya se corresponde con la de un diccionario. Además de ello, faltaba modificar el código para que a cada iteración aplicara el cifrado a cada una de las entradas del diccionario e ingresara los datos en la base. En cada iteración se hace una limpieza de las variables empleadas para el cifrado, salvo el password, que se vacía hasta el final, todo ello con el fin de evitar que se rompa el cifrado a través de un volcado de memoria. El archivo **config.py** nos proporciona las credenciales para que nuestro script pueda llevar a cabo el vaciado de datos en la base.

```

1 password = getpass()
2 with open('diagnosticos-tratamientos.txt') as f:
3     data = f.read()
4     datos = ast.literal_eval(data)
5     for indice in range(len(datos)):
6         name = datos[indice]['name']
7         diagnosis = datos[indice]['diagnosis']
8         treatment = datos[indice]['treatment']
9
10    # El algoritmo de derivación de llaves PBKDF2 necesita una salt,
11    # por lo que generamos una secuencia pseudoaleatoria de 16 bytes.
12    passwordSalt = get_random_bytes(16)
13
14    # En esta práctica emplearemos AES-256, por lo que necesitamos
15    # que el algoritmo de derivación de llaves PBKDF2 nos proporcione
16    # una llave 256 bits (32 bytes).
17    key = PBKDF2(password, passwordSalt, 32, count=1000000, hmac_hash_module=SHA512)
18
19    # Genera nonces nuevos para cada cifrado
20    diagnosis_nonce = get_random_bytes(8)
21    treatment_nonce = get_random_bytes(8)
22    diag_aes = AES.new(key, AES.MODE_CTR, nonce=diagnosis_nonce)
23    treat_aes = AES.new(key, AES.MODE_CTR, nonce=treatment_nonce)
24

```

```

25     # Cifra los campos considerados sensibles
26     diagnosis_ciphertext = diag_aes.encrypt(bytes(diagnosis, 'utf-8'))
27     treatment_ciphertext = treat_aes.encrypt(bytes(treatment, 'utf-8'))
28
29     # Codifica en base 64 tanto la passwordSalt como el criptograma
30     passwordSalt_enc = b64encode(passwordSalt)
31     diagnosis_ciphertext_enc = b64encode(diagnosis_ciphertext)
32     treatment_ciphertext_enc = b64encode(treatment_ciphertext)
33     diagnosis_nonce_enc = b64encode(diagnosis_nonce)
34     treatment_nonce_enc = b64encode(treatment_nonce)
35
36     mydb = None
37     # Guardar los datos en una base de datos relacional
38     try:
39         # Leemos las credenciales para la conexión del archivo config.py
40         # El cifrado de la conexión se realizará en otra práctica
41         mydb = MySQLdb.connect(user=config.user, password=config.password, database=config
.dbname)
42         cursor = mydb.cursor()
43
44         # Ejecutar la consulta SQL para insertar el paciente en la base de datos
45         insert_query = """ INSERT INTO expediente (nombre, diagnostico, tratamiento,
passwordSalt, diag_nonce, treat_nonce)
46                             VALUES (%s,%s,%s,%s,%s,%s)"""
47         record_to_insert = (name, diagnosis_ciphertext_enc, treatment_ciphertext_enc,
passwordSalt_enc, diagnosis_nonce_enc, treatment_nonce_enc)
48         cursor.execute(insert_query, record_to_insert)
49
50         mydb.commit()
51         print("Records inserted successfully")
52
53     except Exception as err:
54         print(f"\nSomething went wrong: {err}")
55         sys.exit()
56
57     finally:
58         if mydb:
59             cursor.close()
60             mydb.close()
61             print("DBMS connection is closed")
62
63
64     # Sobrescribir el contenido de las variables para evitar que se
65     # puedan obtener los datos a través de un volcado de memoria RAM
66     clearmem(key)
67     clearmem(diagnosis)
68     clearmem(treatment)
69
70     clearmem(password)

```

Para correrlo basta con ejecutar el comando `python cifradoAESCTR.py` que nos pedirá la contraseña para cifrar los datos, en este caso “debian12” y nos irá mostrando como se ingresan los datos hasta que concluya. Podemos visualizar la información cifrada ejecutando una consulta en nuestra base, para ello ingresamos a Maria DB con nuestro usuario creado para este fin, ello lo logramos a través del comando `mariadb -user nuevousuario -p` que nos pedirá nuestra contraseña. Una vez dentro del sistema, ingresamos los comandos:

```

1 USE hospital;
2 SELECT * FROM expediente;

```

Que nos entregaran los siguientes resultados:

```
MariaDB [hospital]> select * from expediente;
```

id	nombre	diagnostico	passwordSalt	diag_nonce	treat_nonce	tratamiento
1	Gary Delgado	HAAV/8nmwLLvRzrOG8f5o6t2b/mkgDMwMDX3/9Sh30hu4bB1TJ4JrfpAjgU=	IzcT4z6D0zZgYQEyw56W/g==	RWntiUNFOio=	wsdLALFNGNo=	+4QVULm3xmVGY+wq1rd453h5W0LLrjHMTwM3h4=
2	Russell Lang	Dhq741xGC0eY57HIs2mpK7BoUnjW//3Vs4J4	DKVxfnktsy8M225awMv1Kg==	noFwp3DuW48=	Hc0u9ExsiCk=	ZKBXwtLKwqV/+6oXN0b+Khr3h3kIEwNLAYx0iC/2bQ==
3	Paul Best	Sj5TrLsr9P08avKuXm7Jyihmn7YM8MncrZMbyyAQt33zjF1+pCm8Ga0/sA==	DJLShhehM9kdAyzx+XJdRg==	o2ZNMV+pGGA=	t4D1LUfb/eY=	kQujbEY9wBgCVit0VY8bGa7hSk0u
4	Angelica Griffith	pR7jlyeClbaWi6PLE/fd0jh8vieEveMpkPg+Snp1oQSFqPE9QHxV	pKtBd4a/KEDin+au2yLzcQ==	hcAkt8z3gnQ=	xahMwejQ2rc=	/tCNLMVVvuAyavCyfoZn1MQ1SiPiovfG/sZhIw9U0q2Ub3SgkZv9kpX
5	Barbara Bailey	zA0dpLD9teiX5vBdSycAdizB//ThUNV3dmfB4ar2COhZ8Swa5k=	edh0//pGeoTCL/xQ66apxg==	xrkZAh6vdnI=	mPHFSC+4Pho=	X53V7+ZUDJ8JhS+h0JQJwR261C/ogGZX0eb6J5gK5g36tkb7hBUQ24N

Figura 4: Contenido de la tabla **expediente** de la base de datos **hospital** con el contenido cifrado

2.4. Creación del script para el descifrado AES en modo CTR

Crearemos ahora un nuevo script en python que nos ayude a recuperar todos los datos guardados en la base de datos **hospital.expediente** dado un nombre de algún paciente en particular y nos muestre en terminal el diagnóstico y tratamiento, así como el nombre del paciente en texto plano. A este archivo lo nombraremos como *descifradoAES_CTR.py* y estará ubicado dentro de la carpeta *cifradoAES_CTR*.

Para el descifrado del diagnóstico y tratamiento de cualquier paciente, se necesita recuperar el salt y nonces que se ocuparon para su cifrado, ya que debemos utilizar exactamente la misma llave para descifrarlos.

Para ello, realizaremos el proceso inverso del cifrado que se realizó en el archivo **cifradoAES_CTR.py**. Primero, nos conectaremos a la base de datos con las correspondientes credenciales otorgadas por el archivo **config.py** y capturamos toda la información que haya en la base de datos con relación al nombre del paciente, dada por el usuario desde terminal. Si lo encuentra continuamos, en caso contrario damos aviso que no se encontró ningún paciente con ese nombre en la base de datos. Si todo sale bien, entonces el programa consigue un conjunto con toda la información y la guarda en la variable `datos_paciente`, es decir, en orden devuelve el nombre del paciente (el único en texto plano), luego le sigue, su diagnóstico y tratamiento cifrados, la `passwordSalt`, el nonce de diagnóstico y al final, el nonce de tratamiento.

Lo anterior descrito, ocurre en la siguiente parte de código

```
1 # Conectar a la base de datos
2 mydb = MySQLdb.connect(user=config.user, password=config.password, database=config.dbname)
3 cursor = mydb.cursor()
4
5 # Ejecutar consulta SQL para obtener los datos del paciente
6 consulta_sql = """SELECT diagnostico, tratamiento, passwordSalt, diag_nonce, treat_nonce
7 FROM expediente WHERE nombre = '%s'"""
8
9 cursor.execute(consulta_sql, (nombre,))
10 datos_paciente = cursor.fetchone() #Recuperamos el primer resultado de la consulta
```

Como el último paso del cifrado AES en CTR, fue el codificado en base 64, lo decodificaremos igual en base 64. Esto lo aplicaremos a todos los parámetros recuperados. Antes se aplicó la función **encode()** para que trabajase con una secuencia de bytes. Es decir, se realiza a continuación lo siguiente

```
1 #Aplicamos encode a cada parametro
2 #recuperado para trabajar con una secuencia de bytes
3 diagnostico_base64 = datos_paciente[0].encode()
4 tratamiento_base64 = datos_paciente[1].encode()
5 passwordSalt_base64 = datos_paciente[2].encode()
6 diag_nonce_base64 = datos_paciente[3].encode()
7 trat_nonce_base64 = datos_paciente[4].encode()
8
9 #Ejecutaremos el proceso inverso de encrypt
10 #Primero decodificamos de base 64
11 diagnostico = base64.b64decode(diagnostico_base64)
12 tratamiento = base64.b64decode(tratamiento_base64)
13 passwordSalt = base64.b64decode(passwordSalt_base64)
14 diag_nonce = base64.b64decode(diag_nonce_base64)
15 trat_nonce = base64.b64decode(trat_nonce_base64)
```

Continuamos y ahora le preguntamos al usuario desde terminal, la contraseña que se utilizó para el cifrado. La contraseña debe ser exactamente la misma que se empleó para el cifrado, si el usuario no coloca la contraseña correctamente, entonces no podrá descifrar la información. De igual forma, se utilizó la función `getpass()`. Como ya hemos decodificado las variables `passwordSalt` y obtenido la contraseña que hemos guardado en la variable `password`, procedemos y derivamos nuestra key nuevamente. Para los demás parámetros como la longitud de la llave, el número de iteraciones y el algoritmo hash SHA-512, utilizaremos los mismos que se ocuparon para su cifrado.

```
1 #Utilizamos la misma password dada por el usuario en
2 #CifradoAES_CTR.py
3 password = getpass()
4 key = PBKDF2(password, passwordSalt, 32, count=1000000, hmac_hash_module=SHA512)
```

Ahora, crearemos nuevos objetos de cifrado AES en modo CTR ya que tenemos la key, sabemos el modo de cifrado y hemos decodificado a la variable `diag_nonce` y `trat_nonce`

```
1 diag_aes = AES.new(key, AES.MODE_CTR, nonce=diag_nonce)
2 treat_aes = AES.new(key, AES.MODE_CTR, nonce=trat_nonce)
```

Finalmente, desciframos los datos cifrado y luego los convertimos de bytes a cadena de texto utf-8 para mostrarlos correctamente en terminal

```
1 diag_descifrado = diag_aes.decrypt(diagnostico)
2 diag_descifrado_texto = diag_descifrado.decode('utf-8')
3 trat_descifrado = treat_aes.decrypt(tratamiento)
4 trat_descifrado_texto = trat_descifrado.decode('utf-8')
```

En conclusión, nuestro script .py que nos ayuda con el descifrado dado un nombre de un paciente en particular nos devuelve su diagnostico y tratamiento en texto plano (después de haber sido cifrados con AES en modo CTR) quedo de la siguiente manera

```
1 import MySQLdb
2 import config
3 from Cryptodome.Cipher import AES
4 from Cryptodome.Protocol.KDF import PBKDF2
5 import base64
6 from base64 import b64decode
7 from Cryptodome.Hash import SHA512
8 from getpass import getpass
9
10 def obtener_datos_paciente(nombre):
11     try:
12         # Conectar a la base de datos
13         mydb = MySQLdb.connect(user=config.user, password=config.password, database=config.
14         dbname)
15         cursor = mydb.cursor()
16
17         # Ejecutar consulta SQL para obtener los datos del paciente
18         consulta_sql = """SELECT diagnostico, tratamiento, passwordSalt, diag_nonce,
19         treat_nonce FROM expediente WHERE nombre = %s"""
20
21         cursor.execute(consulta_sql, (nombre,))
22         datos_paciente = cursor.fetchone() #Recuperamos el primer resultado de la consulta
23
24         #Si logramos obtener de forma exitosa, los datos del paciente
25         #en cuestion
26         if datos_paciente:
27
28             diagnostico_base64 = datos_paciente[0].encode()
29             tratamiento_base64 = datos_paciente[1].encode()
30             passwordSalt_base64 = datos_paciente[2].encode()
31             diag_nonce_base64 = datos_paciente[3].encode()
32             trat_nonce_base64 = datos_paciente[4].encode()
33
34             #Ejecutaremos el proceso inverso de encrypt
35             #Primero decodificamos de base 64
36             diagnostico = base64.b64decode(diagnostico_base64)
37             tratamiento = base64.b64decode(tratamiento_base64)
```



```

37     passwordSalt = base64.b64decode(passwordSalt_base64)
38     diag_nonce = base64.b64decode(diag_nonce_base64)
39     trat_nonce = base64.b64decode(trat_nonce_base64)
40
41     #Utilizamos la misma password dada por el usuario en
42     #CifradoAES_CTR.py
43     password = getpass()
44     key = PBKDF2(password, passwordSalt, 32, count=1000000, hmac_hash_module=SHA512)
45
46     # Descifrar los datos utilizando AES
47     diag_aes = AES.new(key, AES.MODE_CTR, nonce=diag_nonce)
48     treat_aes = AES.new(key, AES.MODE_CTR, nonce=trat_nonce)
49
50     diag_decifrado = diag_aes.decrypt(diagnostico)
51     diag_decifrado_texto = diag_decifrado.decode('utf-8')
52     trat_decifrado = treat_aes.decrypt(tratamiento)
53     trat_decifrado_texto = trat_decifrado.decode('utf-8')
54
55     return diag_decifrado_texto, trat_decifrado_texto
56 else:
57     print(f"\nNo hay ningun paciente con nombre:{nombre} en la base de datos. Intenta
de nuevo.")
58     return None, None
59
60 except Exception as e:
61     print(f"Error: {e}")
62     return None, None
63
64 finally:
65     if mydb:
66         #Cerramos el cursor y la conexion a la base de datos
67         cursor.close()
68         mydb.close()
69
70 def main():
71     nombre_paciente = input("Ingrese el nombre del paciente: ")
72
73     diagnostico, tratamiento = obtener_datos_paciente(nombre_paciente)
74     print("\n**Nombre del paciente: ", nombre_paciente)
75     print("**Diagnostico descifrado:", diagnostico)
76     print("**Tratamiento descifrado:", tratamiento)
77
78 if __name__ == "__main__":
79     main()

```

Además se le añadió try catch y manejo de errores para que le señale al usuario cuando ingreso un nombre que no esta en la base de datos o cuando ocurra un error con la conexión a la misma. Este archivo **descifradoAES_CTR.py** se encuentra dentro de la carpeta **cifradoAES_CTR**.


```

gabriela@debian12:~/Descargas/practica9-ultimoCambio/practica9/CifradoAES-CTR$ python3
descifradoAES_CTR.py
Ingrese el nombre del paciente: Robert Taylor
Password:

**Nombre del paciente: Robert Taylor
**Diagnóstico descifrado: Coach no store wear her around.
**Tratamiento descifrado: Maybe discover instead late.
gabriela@debian12:~/Descargas/practica9-ultimoCambio/practica9/CifradoAES-CTR$ python3
descifradoAES_CTR.py
Ingrese el nombre del paciente: kjebfw

No hay ningun paciente con nombre:kjebfw en la base de datos. Intenta de nuevo.

**Nombre del paciente: kjebfw
**Diagnóstico descifrado: None
**Tratamiento descifrado: None

```

Figura 5: Descifrado de los datos del paciente Robert Taylor

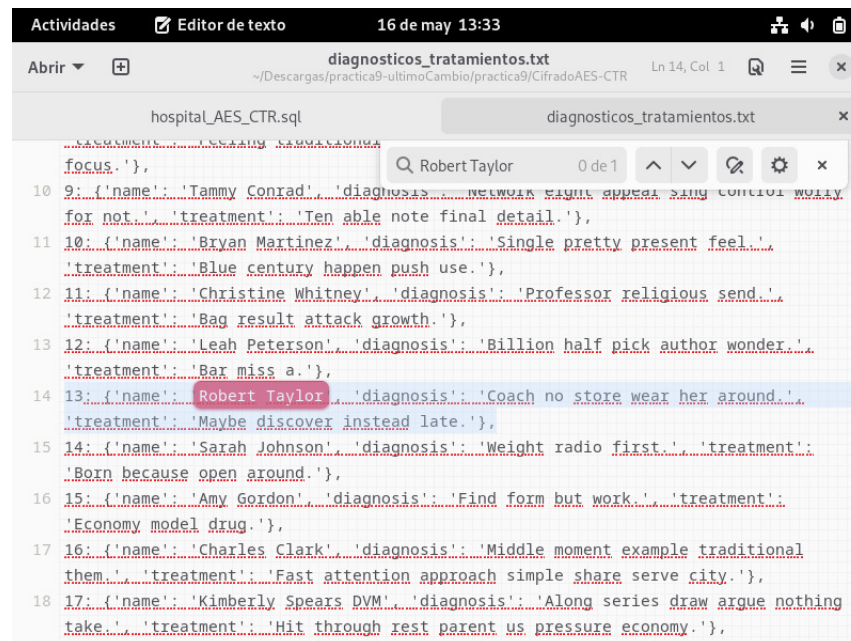
```

MariaDB [hospital]> select * from expediente where nombre = 'Robert Taylor';
+-----+-----+-----+-----+-----+
| id | nombre | diagnostico | tratamiento | passwordSalt | | |
|---|---|---|---|---|---|---|
| 14 | Robert Taylor | suyUdBbmy43Z71jQ+s015B0Gu7TePBsKsJWhMAXbg== | H1AJM/bzk9QfgmnpzF+Z9mNnv8+D1mGoF1uyCA== | yxS87mnFNfpu3CyCIoHxJQ== | 7+/Wt205Trs= | hK0TH8IM1ps= |
+-----+-----+-----+-----+-----+
1 row in set (0.020 sec)

MariaDB [hospital]>

```

Figura 6: Datos del paciente Robert Taylor en la base de datos **hospital**



```

focus.'},
9: {'name': 'Tammy Conrad', 'diagnosis': 'Network eight appeal sing control wuay
for not.', 'treatment': 'Ten able note final detail.'},
10: {'name': 'Bryan Martinez', 'diagnosis': 'Single pretty present feel.',
'treatment': 'Blue century happen push use.'},
11: {'name': 'Christine Whitney', 'diagnosis': 'Professor religious send.',
'treatment': 'Bag result attack growth.'},
12: {'name': 'Leah Peterson', 'diagnosis': 'Billion half pick author wonder.',
'treatment': 'Bar miss a.'},
13: {'name': 'Robert Taylor', 'diagnosis': 'Coach no store wear her around.',
'treatment': 'Maybe discover instead late.'},
14: {'name': 'Sarah Johnson', 'diagnosis': 'Weight radio first.', 'treatment':
'Born because open around.'},
15: {'name': 'Amy Gordon', 'diagnosis': 'Find form but work.', 'treatment':
'Economy model drug.'},
16: {'name': 'Charles Clark', 'diagnosis': 'Middle moment example traditional
them.', 'treatment': 'Fast attention approach simple share serve city.'},
17: {'name': 'Kimberly Spears DVM', 'diagnosis': 'Along series draw argue nothing
take.', 'treatment': 'Hit through rest parent us pressure economy.'},

```

Figura 7: Datos del paciente Robert Taylor en el archivo **diagnosticos_tratamientos.txt**

2.5. Creación del dump de la base de datos de hospital

Creamos el dump con el siguiente comando sql y el archivo generado lo guardamos en dentro de la carpeta **cifradoAES-CTR/dump**.

```

1 mariadb-dump -u nuevousuario -p hospital > hospital_AES_CTR.sql

```

2.6. Cifrado AES en modo GCM y Scrypt en el Proceso de Cifrado

En este paso, adaptamos el script para que utilice GCM (Galois/Counter Mode) para el cifrado y Scrypt para la derivación de claves. La implementación de Scrypt se realiza con el siguiente formato:

```
key = Scrypt(password, salt, N, r, p, derived-key-len)
scrypt(password, kdfSalt, N=16384, r=8, p=1, buflen=32)
```

Derivación de la Clave con Scrypt

Durante el proceso de cifrado, utilizamos la función Scrypt KDF (Key Derivation Function) con parámetros fijos para obtener una clave secreta a partir de una contraseña. El parámetro salt utilizado en la derivación de claves se genera aleatoriamente y se almacena junto con el mensaje cifrado. Este salt será necesario durante el proceso de descifrado para asegurar que la misma clave pueda ser derivada a partir de la misma contraseña.

2.6.1. Creación de una nueva base de datos para el cifrado AES GCM

Crearemos una NUEVA base de datos llamada **hospital2** y una tabla llamada expediente2 con los siguientes campos:

ID del paciente

Nombre del paciente (en texto claro)

Diagnóstico (cifrado)

Tratamiento médico (cifrado)

Para ello utilizaremos, el mismo esquema. Le cambiaremos el nombre de la base y el de la tabla.

```
1 CREATE DATABASE IF NOT EXISTS hospital;
2 USE hospital;
3
4 CREATE TABLE IF NOT EXISTS expediente (
5     id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
6     nombre VARCHAR(255) NOT NULL,
7     diagnostico VARCHAR(450) NOT NULL,
8     tratamiento VARCHAR(450) NOT NULL,
9     passwordSalt VARCHAR(25) NOT NULL,
10    diag_nonce VARCHAR(12) NOT NULL,
11    treat_nonce VARCHAR(12) NOT NULL
12 );
```

Field	Type	Null	Key	Default	Extra
id	int(6) unsigned	NO	PRI	NULL	auto_increment
nombre	varchar(255)	NO		NULL	
diagnostico	varchar(450)	NO		NULL	
tratamiento	varchar(450)	NO		NULL	
passwordSalt	varchar(25)	NO		NULL	
diag_nonce	varchar(12)	NO		NULL	
treat_nonce	varchar(12)	NO		NULL	

Figura 8: Campos para la base de datos

2.6.2. Lectura del Archivo de Texto:

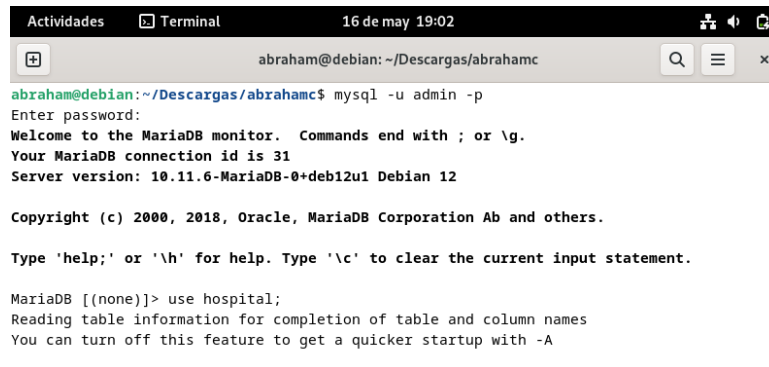
Leeremos un archivo de texto (.txt). Este archivo contiene la información del paciente. Este archivo incluirá datos sensibles que necesitamos cifrar, específicamente el diagnóstico y el tratamiento médico.

2.6.3. Cifrado de Datos Sensibles:

Utilizaremos nuestro script para cifrar los datos sensibles del archivo de texto. El cifrado se realizará utilizando GCM para asegurar la confidencialidad e integridad de los datos.

2.6.4. Almacenamiento en la Base de Datos:

Insertaremos los datos cifrados en la base de datos hospital2, dentro de la tabla expediente2. El nombre del paciente se almacenará en texto claro, mientras que el diagnóstico y el tratamiento médico estarán cifrados.



```
Abraham@debian: ~/Descargas/abrahamc
abraham@debian:~/Descargas/abrahamc$ mysql -u admin -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 31
Server version: 10.11.6-MariaDB-0+deb12u1 Debian 12

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> use hospital;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
```

Figura 9: Abrimos la bd desde la terminal.

2.6.5. Verificación:

Finalmente, verificaremos que la base de datos contiene los datos del paciente correctamente cifrados. Solo el nombre del paciente será visible en texto claro.



```
MariaDB [hospital]> select * from expediente;
+-----+-----+-----+-----+-----+-----+-----+-----+
| id | nombre | diagnostico | passwordSalt | diag_nonce | treat_nonce | diag_tag | tre |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | John Connor | C/2Bf8GRA0WLF5jGxP9KhaD3NLYpW6aX58HA | NeMmrJg5n3l63mazLP1wdwJGZbmm8w== | gjt4EcMbuiE= | X368liaXkSk= | FuGV5JswIA+w | gEilUp09Ug== |
| 2 | Gary Delgado | Next pay deep cold pretty attention suggest. | Well stuff significant mouth. | ZDIzYWM0NGU5YzVhNmMzZDhmOWVlOGM= | Q6sQ8rN4kRGNmBNf | HBMj9KOp1bvLwVfF | YWExZjJkM2Y0 |
| 3 | Russell Lang | Break spend lav east exist. | ZDIzYWM0NGU5YzVhNmMzZDhmOWVlOGM= | Q6sQ8rN4kRGNmBNf | HBMj9KOp1bvLwVfF | YWExZjJkM2Y0 | YWExZjJkM2Y0 |
```

Figura 10: Los datos se han cifrado correctamente.

Nuestro script que realiza el cifrado anterior lo hemos llamado **cifradoAES_GCM.py** y se encuentra dentro de la otra carpeta **cifradoAES_GCM**. El cual, quedo de la siguiente manera

```
1 from getpass import getpass
2 from base64 import b64encode
3
4 from os import urandom
5 from getpass import getpass
6 from base64 import b64encode
7 #from SecureString import clearmem
8 from Cryptodome.Protocol.KDF import PBKDF2
9 #from Crypto.Protocol.KDF import PBKDF2
10 from Cryptodome.Hash import SHA512
11 from Cryptodome.Random import get_random_bytes
12 from Cryptodome.Cipher import AES
13 from cryptography.hazmat.backends import default_backend
14 from cryptography.hazmat.primitives.kdf.scrypt import Scrypt
15 from cryptography.hazmat.primitives.kdf.scrypt import Scrypt
```

```

16
17 from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
18 import mysql.connector
19 import config
20 import sys
21
22 import json
23 import ast
24
25 # Se obtiene la contraseña para cifrar los datos
26 password = getpass()
27
28 # La idea es que estos datos vengan de otro lado como parte
29 # de una app más completa.
30 name = 'John Connor'
31 diagnosis = bytes('Heridas por ataque de T-800', 'utf-8')
32 treatment = bytes('Paracetamol cada 8 hrs', 'utf-8')
33
34 # Datos de entrada
35 passwd = b'p@$Sw0rD~7'
36 salt1 = b'aa1f2d3f4d23ac44e9c5a6c3d8f9ee8c'
37
38 # Utilizamos scrypt para derivar una clave
39 kdf_scrypt = Scrypt(salt=salt1, length=32, n=16384, r=8, p=1, backend=default_backend())
40 key = kdf_scrypt.derive(passwd)
41
42 print("Derived key using scrypt:", key.hex())
43
44 # Se generan nonces nuevos para GCM
45 diagnosis_nonce = urandom(12)
46 treatment_nonce = urandom(12)
47
48 # Se cifran los datos utilizando AES en modo GCM
49 cipher = Cipher(algorithms.AES(key), modes.GCM(diagnosis_nonce), backend=default_backend())
50 encryptor = cipher.encryptor()
51 diagnosis_ciphertext = encryptor.update(diagnosis) + encryptor.finalize()
52 diagnosis_tag = encryptor.tag
53
54 cipher = Cipher(algorithms.AES(key), modes.GCM(treatment_nonce), backend=default_backend())
55 encryptor = cipher.encryptor()
56 treatment_ciphertext = encryptor.update(treatment) + encryptor.finalize()
57 treatment_tag = encryptor.tag
58
59
60 # Se codifica en base64 tanto la salt como el criptograma y el tag para guardarlos en la base
61 # de datos
62 salt = b64encode(salt1)
63 diagnosis_ciphertext = b64encode(diagnosis_ciphertext)
64 treatment_ciphertext = b64encode(treatment_ciphertext)
65 diagnosis_nonce = b64encode(diagnosis_nonce)
66 treatment_nonce = b64encode(treatment_nonce)
67 diagnosis_tag = b64encode(diagnosis_tag)
68 treatment_tag = b64encode(treatment_tag)
69
70 # Print de la información cifrada y codificada
71 print('Diagnosis:', diagnosis)
72 print('Medical treatment:', treatment)
73 print('PasswordSalt:', salt)
74 print('AES encryption key:', key.hex())
75 print(f"Nonces:- diagnosis:{diagnosis_nonce.decode()}; treatment:{treatment_nonce.decode()}")
76 print('Diagnosis encrypted:', diagnosis_ciphertext)
77 print('Treatment encrypted:', treatment_ciphertext)
78
79 # Definimos una función para guardar la información en la base de datos
80 # Definir la función para leer los registros del archivo
81 def leer_registros_desde_archivo(archivo):
82     try:
83         with open(archivo, 'r') as file:
84             content = file.read()

```

```

84         registros = eval(content) # Utilizamos eval para interpretar el contenido como un
            diccionario
85         return registros
86     except Exception as e:
87         print("Error al leer el archivo:", e)
88         return None
89
90 # Definir la función para procesar y guardar los registros en la base de datos
91 def procesar_y_guardar_registros(registros):
92     if registros:
93         for key, value in registros.items():
94             try:
95                 # Establecer la conexión a la base de datos
96                 mariadb = mysql.connector.connect(user=config.user, password=config.password,
97                 database=config.dbname)
98                 cursor = mariadb.cursor()
99
100                query = "INSERT INTO expediente2 (nombre, diagnostico, tratamiento,
101                passwordSalt, diag_nonce, treat_nonce) VALUES (%s, %s, %s, %s, %s, %s)"
102                cursor.execute(query, (value['name'], value['diagnosis'], value['treatment'],
103                salt, diagnosis_nonce.decode(), treatment_nonce.decode() ))
104
105                # Guardar los cambios en la base de datos
106                mariadb.commit()
107                print("Registro guardado exitosamente en la base de datos.")
108
109            except Exception as e:
110                print("Error al guardar el registro en la base de datos:", e)
111                # En caso de error, revertir los cambios
112                mariadb.rollback()
113
114            finally:
115                # Cerrar la conexión a la base de datos
116                mariadb.close()
117
118 # Leer los registros del archivo
119 registros = leer_registros_desde_archivo("diagnosticos_tratamientos.txt")
120
121 # Procesar y guardar los registros en la base de datos
122 procesar_y_guardar_registros(registros)
123
124 del password
125 del key

```

2.7. Descifrado AES en modo GCM

Para ello, después de llenado de la tabla de la base de datos **hospital2.expediente2** modificamos el tipo de dato de las columnas diagnostico y tratamiento, y agregamos otras dos, *diag_tag* y *treat_tag* con los siguientes comandos sql

```

1 ALTER TABLE expediente2
2 MODIFY COLUMN diagnostico VARBINARY(1000) NOT NULL,
3 MODIFY COLUMN tratamiento VARBINARY(1000) NOT NULL,
4 ADD COLUMN diag_tag VARCHAR(16) NOT NULL,
5 ADD COLUMN treat_tag VARCHAR(16) NOT NULL;

```

Posteriormente creamos ahora un nuevo script en python que nos ayude a recuperar los todos los datos guardados en la base de datos **hospital2.expediente2** dado un nombre de algún paciente en particular y nos muestre en terminal el diagnostico y tratamiento, así como el nombre del paciente en texto plano. A este archivo lo nombraremos como *descifradoAES_GCM.py* y estará ubicado dentro de la carpeta *cifradoAES_GCM*.

La estructura del código en general es:

Se importan varias bibliotecas necesarias para el, descifrado, manejo de bases de datos y la gestión de contraseñas.

```

1 from getpass import getpass
2 from base64 import b64decode, b64encode
3 from cryptography.hazmat.backends import default_backend

```

```

4 from cryptography.hazmat.primitives.kdf.scrypt import Scrypt
5 from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
6 import mysql.connector
7 import config

```

La función **derive_key** utiliza el algoritmo Scrypt para derivar una clave de cifrado a partir de una contraseña y una sal (salt). Ya que Scrypt es un algoritmo de derivación de clave que es resistente a ataques de fuerza bruta.

```

1 def derive_key(password, salt):
2     kdf = Scrypt(salt=salt, length=32, n=16384, r=8, p=1, backend=default_backend())
3     return kdf.derive(password)

```

Además se tiene la función **buscar_paciente** que podría considerarse la función principal de éste script ya que esta función se encarga de buscar y descifrar los datos de un paciente en la base de datos. Incluye varios pasos:

- Conexión a la base de datos.
- Recuperación de datos cifrados y metadatos (salt, nonces, tags) desde la base de datos.
- Decodificación de los datos desde Base64
- Derivación de la clave de cifrado.
- Descifrado de los datos utilizando AES-GCM.

```

1 # Funci n para buscar y descifrar los datos de un paciente en la base de datos
2 def buscar_paciente(nombre, password):
3     try:
4         # Establecer conexi n con la base de datos
5         connection = mysql.connector.connect(user=config.user, password=config.password,
6         database=config.dbname)
7         cursor = connection.cursor()
8
9         # Consulta SQL para obtener los datos cifrados del paciente
10        query = "SELECT diagnostico, tratamiento, passwordSalt, diag_nonce, treat_nonce,
11        diag_tag, treat_tag FROM expediente2 WHERE nombre = %s"
12        cursor.execute(query, (nombre,))
13        record = cursor.fetchone()
14
15        if record:
16            # Obtener datos cifrados y metadatos
17            diagnosis_ciphertext, treatment_ciphertext, password_salt, diag_nonce, treat_nonce
18            , diag_tag, treat_tag = record
19
20            # Imprimir los datos recuperados antes de la decodificaci n
21            print("Datos recuperados de la base de datos:")
22            print("Diagn stico: ", diagnosis_ciphertext.decode())
23            print("Tratamiento: ", treatment_ciphertext.decode())
24
25            # Decodificar los datos desde base64
26            try:
27                diagnosis_ciphertext = b64decode(diagnosis_ciphertext)
28                treatment_ciphertext = b64decode(treatment_ciphertext)
29                password_salt = b64decode(password_salt)
30                diag_nonce = b64decode(diag_nonce)
31                treat_nonce = b64decode(treat_nonce)
32                diag_tag = b64decode(diag_tag)
33                treat_tag = b64decode(treat_tag)
34            except Exception as e:
35                return
36
37            # Derivar la clave de cifrado utilizando scrypt y la sal almacenada en la base de
38            datos
39            key = derive_key(password.encode(), password_salt)
40
41            # Descifrar el diagn stico utilizando AES-GCM
42            cipher = Cipher(algorithms.AES(key), modes.GCM(diag_nonce, diag_tag), backend=
43            default_backend())
44            decryptor = cipher.decryptor()

```

```

40         diagnosis = decryptor.update(diagnosis_ciphertext) + decryptor.finalize()
41
42         # Descifrar el tratamiento utilizando AES-GCM
43         cipher = Cipher(algorithms.AES(key), modes.GCM(treat_nonce, treat_tag), backend=
default_backend())
44         decryptor = cipher.decryptor()
45         treatment = decryptor.update(treatment_ciphertext) + decryptor.finalize()
46
47
48     else:
49         print("Paciente no encontrado en la base de datos.")
50
51 except mysql.connector.Error as e:
52     print("Error al buscar paciente en la base de datos:", e)
53
54 except Exception as e:
55     print("Error al descifrar los datos:", e)
56
57 finally:
58     # Cerrar la conexión con la base de datos
59     if connection.is_connected():
60         cursor.close()
61         connection.close()

```

La interacción con el usuario consiste en solicitar el nombre del paciente y la contraseña para descifrar los datos.

Se llama a la función `buscar_paciente` con los datos proporcionados.

```

1     # Solicitar al usuario el nombre del paciente a buscar
2 nombre_paciente = input("Ingrese el nombre del paciente: ")
3
4 # Solicitar la contraseña
5 password = getpass("Ingrese la contraseña para descifrar los datos: ")
6
7 # Buscar y descifrar los datos del paciente
8 buscar_paciente(nombre_paciente, password)

```

Finalmente cabe mencionar el manejo de errores del script para capturar y reportar problemas que puedan surgir durante la conexión a la base de datos, la decodificación de datos, la derivación de la clave y el proceso de descifrado. Esto asegura que cualquier fallo sea reportado adecuadamente, proporcionando información útil para la resolución de problemas.

2.8. Creación del dump de la base de datos de hospital

Creamos el dump con el siguiente comando sql y el archivo generado lo guardamos en dentro de la carpeta `cifradoAES-GCM/dump`.

```

1 mariadb-dump -u nuevousuario -p hospital > hospital2_AES_GCM.sql

```

3. Conclusión

Durante el desarrollo de la practica para cifrar y almacenar información de pacientes utilizando GCM y Crypt, tuvimos algunas complicaciones que fueron resueltas con diferentes estrategias.

Instalación de Paquetes y Problemas de Importación:

PyCryptodome: Uno de los primeros problemas fue la instalación y correcta importación del paquete `pycryptodome`. En el código de referencia, los módulos de este paquete se importaban con nombres diferentes a los esperados. Por ejemplo:

```

from Cryptodome.Protocol.KDF import PBKDF2
from Cryptodome.Hash import SHA512

```

Después de investigar, se encontró que para que funcionara correctamente, los módulos debían ser importados de la siguiente manera:

```
from Crypto.Protocol.KDF import PBKDF2
from Crypto.Hash import SHA512
```

Importación de MySQL:

Otro problema fue la importación del módulo mysql en Python. La terminal arrojaba mensajes de error que no eran claros. Para solucionar esto, se decidió crear un entorno virtual (virtualenv) e instalar todos los paquetes necesarios, incluyendo pycryptodome, MySQLdb y securestring. Esto ayudó a aislar las dependencias y evitar conflictos con otras instalaciones de Python en el sistema.

Conexión y Manejo de la Base de Datos: Por otro lado, tuvimos problema al momento de descifrar GCM ya que no podíamos establecer una conexión segura y correcta con la base de datos MariaDB. Sin embargo ésto lo pudimos solucionar utilizando del módulo mysql.connector para gestionar la conexión y las operaciones con la base de datos. Se implementó manejo de errores para capturar y reportar problemas de conexión.

Nonces y Tags Incorrectos:

Además, con el descifrado el problema que más nos costó solucionar que eran los valores de nonces y tags necesarios para el modo de cifrado AES-GCM ya que no eran correctamente almacenados o recuperados. En la manera que lo solucionamos aseguramos que los nonces y tags se codificaran y decodificaran correctamente en Base64. Debido a que añadimos impresiones de depuración para verificar la correcta recuperación de estos valores y así poder identificar dónde estaba rompiéndose el flujo o estaba la pérdida de información. Finalmente tuvimos fallos en el proceso de descifrado debido a claves incorrectas, datos malformados o metadatos incorrectos. Para lo que implementamos bloques de manejo de excepciones alrededor del proceso de descifrado para capturar y reportar cualquier error, proporcionando información un poco más detallada para la resolución de problemas.

Formato del Archivo de Texto:

Un problemita adicional, aunque menor, fue el formato del archivo de texto que contenía la información del paciente. Los datos aparecían en una sola línea continua, lo que dificultaba su procesamiento, cuando menos para quienes hicieron el cifrado AES-GCM. Para mantener un orden y facilitar la verificación del script, se modificó el archivo de texto para que los datos estuvieran separados por saltos de línea. Esto permitió una lectura y manipulación más sencilla de la información.

A pesar de los inconvenientes antes presentados, esta práctica fue relativamente sencilla y nos ayudo a comprender mejor como el cifrado de datos se puede emplear en un entorno real donde la confidencialidad de la información es uno de los ejes rectores del desarrollo de aplicaciones. Gracias a esta práctica, nos sentimos un poco mejor preparados de cara al mundo laboral y académico, que nos requieren de saber como proteger información sensible de posibles atacantes.

4. Referencias

- Principios y deberes en materia de protección de datos personales.(06 de Enero de 2022). El Banco del Bienestar, Sociedad Nacional de Crédito, Institución de Banca de Desarrollo, (Recuperado el 07 de Mayo de 2024). https://www.gob.mx/cms/uploads/attachment/file/763381/Principios_y_deberes_en_materia_de_Proteccion_de_Datos_Personales.pdf.
- AES Encrypt / Decrypt - Examples(Noviembre 2018). Practical Cryptography for Developers (Recuperado el 07 de Mayo de 2024). <https://cryptobook.nakov.com/symmetric-key-ciphers/aes-encrypt-decrypt-examples>.
- Kenlon,Seth. (28 de marzo de 2022). How to get started with MySQL and MariaDB. RedHat, Inc. website. (Recuperado el 12 de mayo de 2024) <https://www.redhat.com/sysadmin/mysql-mariadb-introduction>.