

UNIVERSIDAD MESOAMERICANA
QUETZALTENANGO
FACULTAD DE INGENIERÍA
INGENIERÍA EN SISTEMAS, INFORMÁTICA Y CIENCIAS DE LA COMPUTACIÓN.



Cifrado y hash de credenciales de conexión a bases de datos con RSA y BCrypt.

JUAN CARLOS NEÍL PALACIOS ESCOBAR

No. de carné 202008022

RONY LISANDRO CARPIO ALVARADO

No. de carné 201908029

JOEL JUAN PABLO GRAMAJO CHAN

No. de carné 202008025

QUETZALTENANGO, Mayo 2023.

RSA

El algoritmo RSA (Rivest-Shamir-Adleman) es un sistema de criptografía de clave pública ampliamente utilizado para el cifrado y la firma digital. Fue inventado en 1977 por Ron Rivest, Adi Shamir y Leonard Adleman.

RSA se basa en la dificultad computacional de factorizar grandes números compuestos en sus factores primos. Su seguridad se basa en la idea de que es muy difícil factorizar números grandes en un tiempo razonable, incluso utilizando computadoras potentes.

Pasos:

1. Generación de claves:

- Seleccionar dos números primos grandes, p y q .
- Calcular el producto $n = p * q$. Este número es conocido como el módulo.
- Calcular la función totient de Euler $\phi(n) = (p-1) * (q-1)$. Esta función cuenta el número de enteros positivos menores que n y coprimos con n .
- Elegir un número e ($1 < e < \phi(n)$) que sea coprimo con $\phi(n)$ (es decir, no tenga factores comunes excepto 1).
- Calcular d como el inverso multiplicativo de e módulo $\phi(n)$. Esto se puede hacer utilizando el algoritmo extendido de Euclides.
- La clave pública consiste en los valores (e, n) , y la clave privada consiste en el valor (d, n) .

2. Cifrado:

- El mensaje original se divide en bloques más pequeños.
- Para cada bloque, se calcula el valor cifrado utilizando la fórmula $c = m^e \bmod n$, donde m es el bloque del mensaje y c es el bloque cifrado.

3. Descifrado:

- Para cada bloque cifrado, se calcula el valor original utilizando la fórmula $m = c^d \bmod n$, donde c es el bloque cifrado y m es el bloque del mensaje original.

La seguridad del algoritmo RSA se basa en la dificultad de factorizar el número n en sus factores primos p y q . Si se conocen los factores, se puede calcular fácilmente la clave privada y descifrar el mensaje. Sin embargo, hasta ahora no se ha encontrado un algoritmo eficiente para factorizar grandes números en tiempo polinómico, lo que hace que RSA sea seguro para su uso práctico.

Es importante destacar que RSA se utiliza no solo para cifrar y descifrar mensajes, sino también para la firma digital. Al firmar digitalmente un mensaje con la clave privada, se puede demostrar la autenticidad y la integridad del remitente, ya que solo el poseedor de la clave privada correspondiente puede generar una firma válida.

BCrypt

BCrypt es una función de hash criptográfica diseñada específicamente para el almacenamiento seguro de contraseñas. A diferencia de los algoritmos de hash estándar como MD5 o SHA-1, bcrypt se caracteriza por ser lento y resistente a ataques de fuerza bruta.

Funcionamiento:

1. Salt (sal): bcrypt utiliza un concepto llamado "salt", que es una cadena aleatoria generada de forma única para cada contraseña. El salt se agrega a la contraseña antes de aplicar la función de hash, lo que garantiza que aunque dos usuarios tengan la misma contraseña, sus hash serán diferentes debido a los salts diferentes.
2. Hashing: bcrypt utiliza una función de hash iterativa que realiza múltiples rondas de un algoritmo de cifrado de clave simétrica, conocido como Blowfish. La cantidad de rondas de cifrado es controlada por un factor de trabajo llamado "cost factor", que determina cuánto tiempo se tarda en generar el hash. Este factor de trabajo se selecciona de manera que el proceso de hashing sea lo suficientemente lento como para dificultar los ataques de fuerza bruta y de diccionario.
3. Verificación de contraseña: Para verificar una contraseña, bcrypt toma la contraseña proporcionada por el usuario, agrega el salt almacenado previamente y genera un hash. Luego, compara este hash con el almacenado en la base de datos. Si coinciden, la contraseña se considera válida.

La ventaja principal de bcrypt es su resistencia a los ataques de fuerza bruta y de diccionario. El tiempo requerido para generar cada hash es configurable, lo que significa que se puede aumentar el costo en el futuro si aumenta la capacidad computacional. Además, al utilizar un salt único para cada contraseña, se evita el uso de tablas de búsqueda precalculadas (rainbow tables) que aceleran los ataques de hash.

Hash con BCrypt

Bcrypt es una función de hash criptográfica utilizada principalmente para el almacenamiento seguro de contraseñas, no para cifrar datos en general.

La función de hash bcrypt es unidireccional, lo que significa que no se puede revertir para obtener la cadena original. Su propósito principal es convertir una contraseña en un valor hash irreconocible, de modo que, al almacenar este hash, no sea posible obtener la contraseña original. Esto es útil para verificar si una contraseña ingresada es válida sin almacenarla en texto plano.

Si estás buscando cifrar una cadena de conexión para asegurar la confidencialidad de los datos durante la transmisión, bcrypt no es el algoritmo adecuado para esa tarea. Para el cifrado de datos, se utilizan algoritmos de cifrado simétrico o asimétrico, como AES o RSA, respectivamente.

Es importante tener en cuenta que el cifrado y el hash son conceptos diferentes. El cifrado implica transformar datos legibles en un formato ilegible (cifrado) y luego restaurarlos a su forma original (descifrado). Por otro lado, la función de hash, como bcrypt, transforma una entrada en un valor de hash irreconocible, pero no puede recuperar la entrada original.

AES

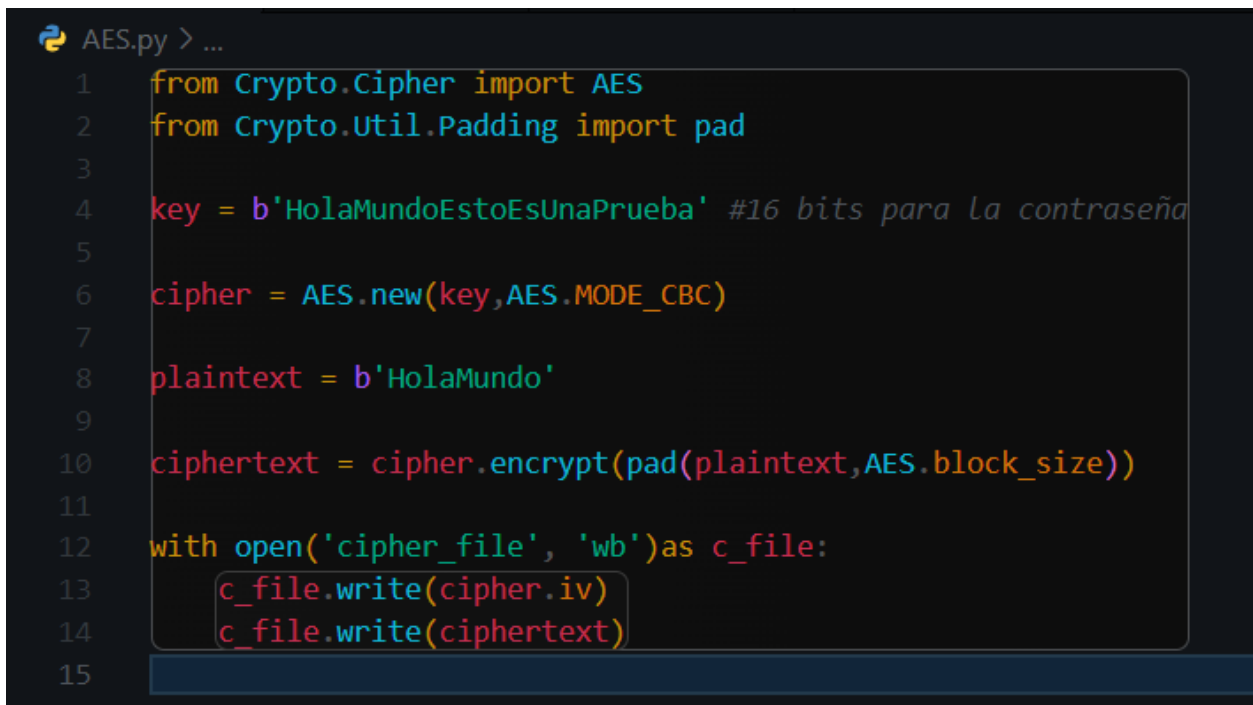
Es un algoritmo de cifrado simétrico ampliamente utilizado en la criptografía, es muy utilizado para sistemas de seguridad.

El cifrado AES se utiliza para proteger la información confidencial, como las credenciales de acceso a la base de datos, durante la transmisión o almacenamiento. En lugar de enviar o almacenar las credenciales en texto plano, se cifran utilizando este algoritmo.

Cuando se utiliza AES para cifrar una cadena de conexión de base de datos, se convierte en una secuencia de caracteres cifrados que no pueden ser fácilmente entendidos o utilizados por alguien que no tenga la clave de cifrado adecuada, lo cual proporciona una capa adicional de seguridad para proteger la información sensible de la base de datos.

El uso de cifrado AES en las cadenas de conexión de bases de datos ayuda a salvaguardar la información confidencial y a prevenir accesos no autorizados o manipulados de los datos.

Cifrado con AES y Pycryptodome

A screenshot of a code editor showing a Python script named 'AES.py'. The script demonstrates AES encryption using the Pycryptodome library. It imports AES and padding from the Crypto module, defines a 16-byte key, creates a cipher object in CBC mode, pads the plaintext 'HolaMundo', and encrypts it. The resulting ciphertext and the IV are then written to a file named 'cipher_file'.

```
AES.py > ...
1  from Crypto.Cipher import AES
2  from Crypto.Util.Padding import pad
3
4  key = b'HolaMundoEstoEsUnaPrueba' #16 bits para la contraseña
5
6  cipher = AES.new(key,AES.MODE_CBC)
7
8  plaintext = b'HolaMundo'
9
10 ciphertext = cipher.encrypt(pad(plaintext,AES.block_size))
11
12 with open('cipher_file', 'wb') as c_file:
13     c_file.write(cipher.iv)
14     c_file.write(ciphertext)
15
```

Importamos las clases que utilizaremos las cuales son AES para el cifrado con lo cual realiza el relleno (padding) el cual será necesario para el bloque de datos.

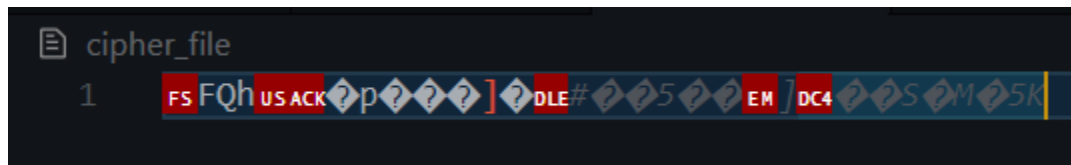
Luego definimos una clave de 16 bytes(128 bits) el cual será utilizado con AES.

Se creó una instancia llamada AES la cual tendrá una clave sobre el cifrado.

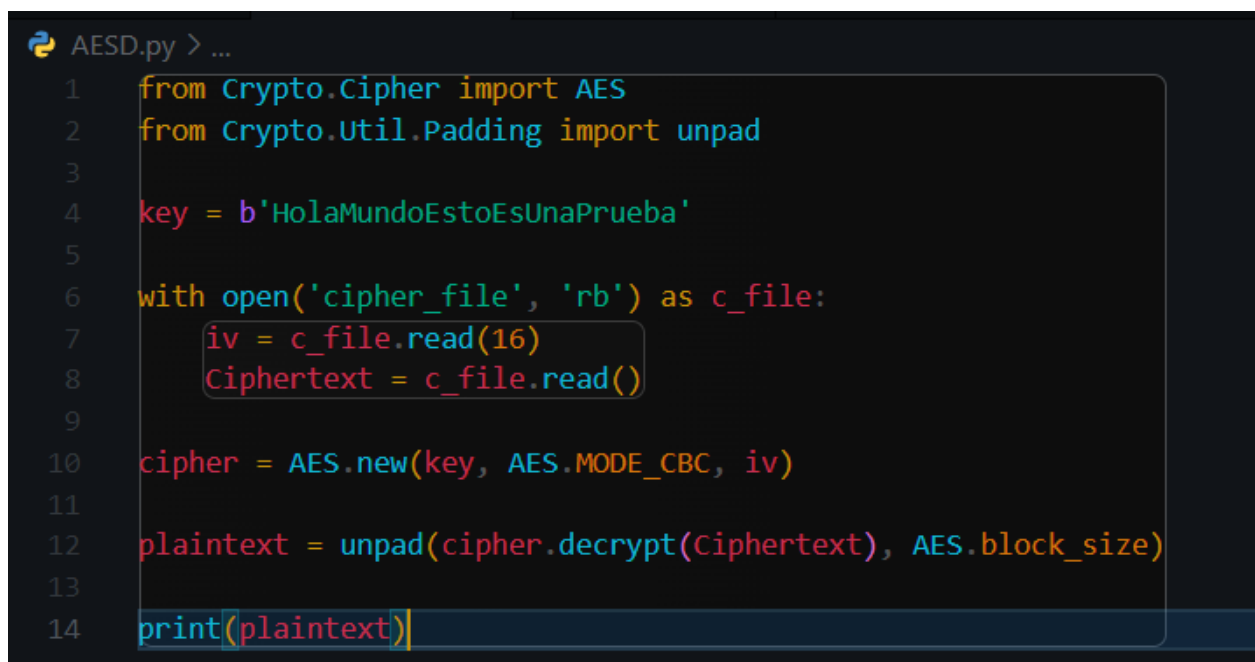
Luego se definió el texto el cual será encriptado con la función plaintext = b'TEXTO PARA AGREGAR'.

El texto creado anteriormente se utiliza para el método encrypt() del objeto cipher con lo cual se le aplica un relleno necesario utilizando el método pad() del módulo crypto.util.padding.

Por último el archivo creará un archivo con el cual escribirá el vector de inicialización en el archivo, lo cual será necesario para descifrar los datos correctamente en el modo CBC y luego se escriben cifrados en el archivo cipher_file.



Descifrado con AES y Pycryptodome



Importamos las clases para realizar el descifrado AES y para eliminar el relleno que nos genera el código anterior (padding).

En esta parte es importante definir la misma key (clave de cifrado utilizada en el archivo anterior) por que si no se utiliza cuando se intente descifrar el archivo lleva un codigo y como no es el mismo código que se generó en el script generará errores, entonces debe ser la misma clave generada anteriormente.

Abriremos el archivo `cipher_file` en modo lectura binaria y se podrá observar el lector de inicialización de 16 bytes del archivo, el cual fue escrito con el texto cifrado, una vez hecho esto veremos el texto cifrado del archivo y podremos comparar si tiene el mismo texto o viceversa.

cipher = AES.new(key, AES.MODE_CBC, iv) con este comando crearemos una instancia AES con la clave del cifrado en modo operación CBC lo cual esta instancia se utilizará para descifrar el texto cifrado.

Pruebas

Cifrado anterior:

```

1  FS FQh US ACK ?p???] ?DLE#??5??EM] DC4 ??S?M?5K

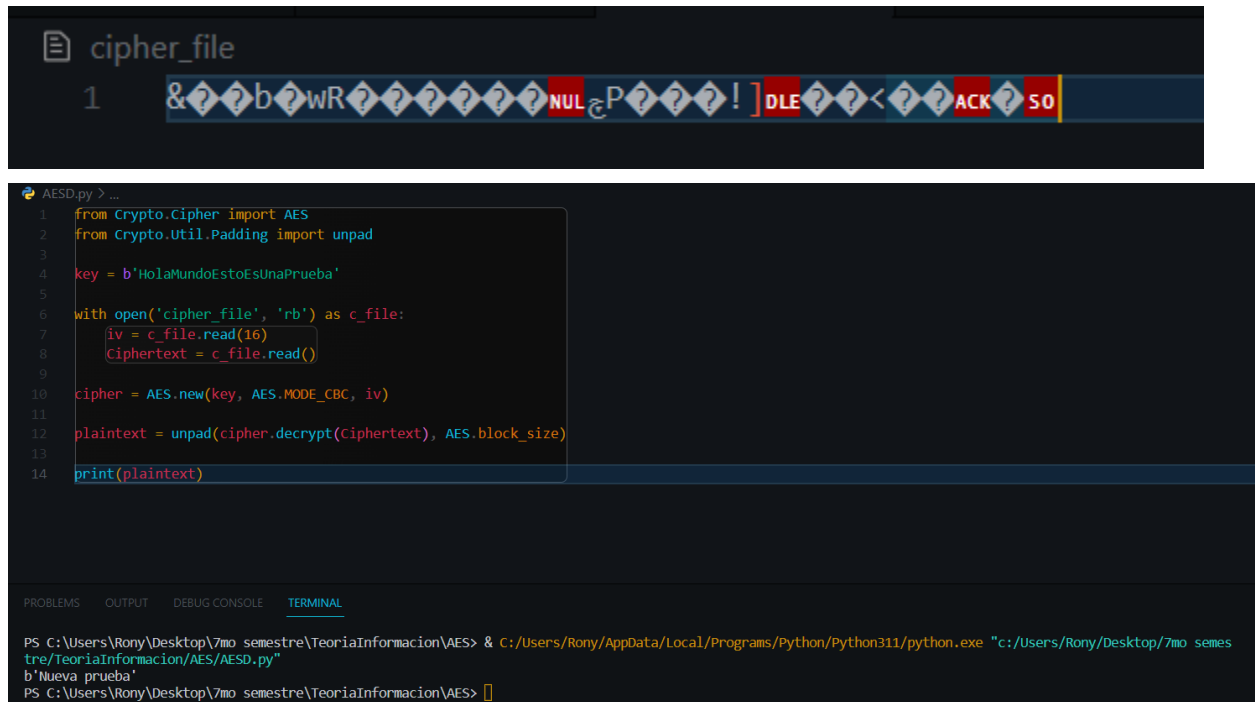
```

```

AES.py > ...
1  from Crypto.Cipher import AES
2  from Crypto.Util.Padding import pad
3
4  key = b'HolaMundoEstoEsUnaPrueba' #16 bits para la contraseña
5
6  cipher = AES.new(key,AES.MODE_CBC)
7
8  plaintext = b'Nueva prueba'
9
10 ciphertext = cipher.encrypt(pad(plaintext,AES.block_size))
11
12 with open('cipher_file', 'wb') as c_file:
13     c_file.write(cipher.iv)
14     c_file.write(ciphertext)
15

```

Nuevo Cifrado:



The image shows a screenshot of a Python IDE with a dark theme. At the top, a file named 'cipher_file' is open, displaying a hex dump of a ciphertext. The hex dump shows various bytes including '&', 'b', 'w', 'R', 'NUL', 'P', '!', 'DLE', '<', 'ACK', and 'SO'. Below the file editor, a Python script named 'AESD.py' is shown. The script imports AES and unpad from the Crypto module, defines a key, reads an IV and ciphertext from the 'cipher_file', and then decrypts the ciphertext using AES in CBC mode. The output of the script is printed in the terminal. The terminal shows the command being executed and the resulting plaintext: 'b'Nueva prueba'.

```
1 from Crypto.Cipher import AES
2 from Crypto.Util.Padding import unpad
3
4 key = b'HolaMundoEstoEsUnaPrueba'
5
6 with open('cipher_file', 'rb') as c_file:
7     iv = c_file.read(16)
8     Ciphertext = c_file.read()
9
10 cipher = AES.new(key, AES.MODE_CBC, iv)
11
12 plaintext = unpad(cipher.decrypt(Ciphertext), AES.block_size)
13
14 print(plaintext)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

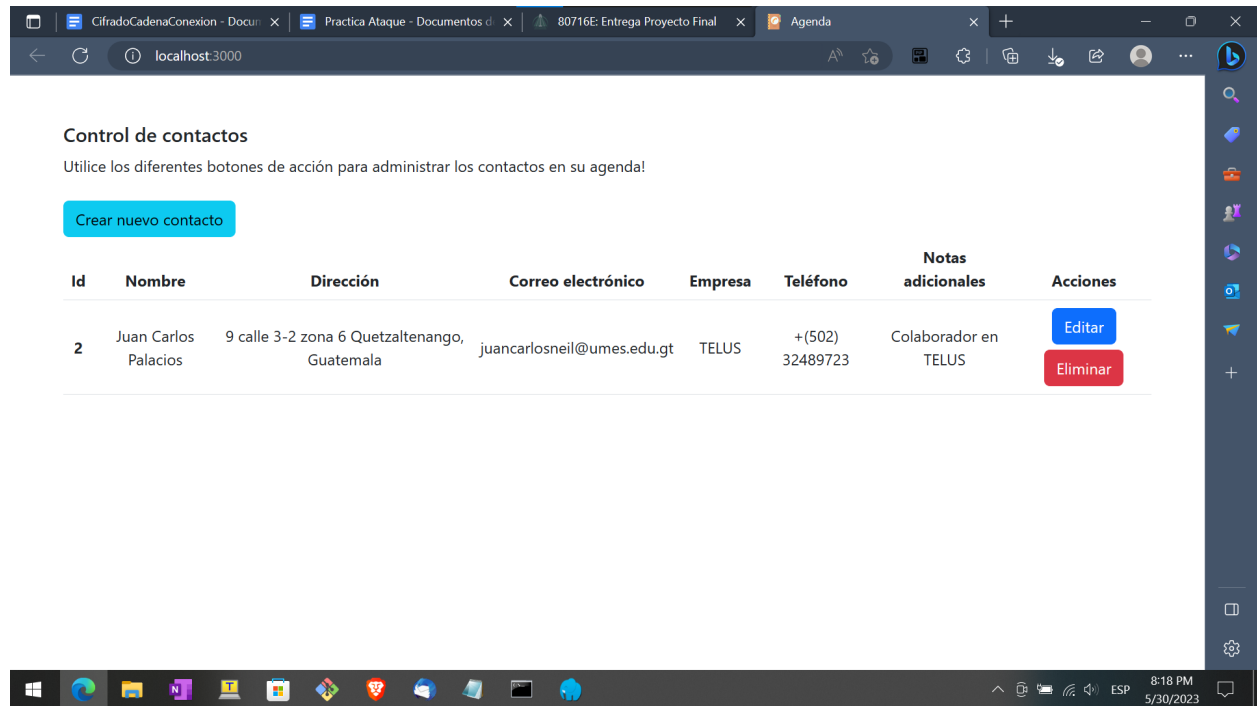
PS C:\Users\Rony\Desktop\7mo semestre\TeoriaInformacion\AES> & C:/Users/Rony/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/Rony/Desktop/7mo semes
tre/TeoriaInformacion/AES/AESD.py"
b'Nueva prueba'
PS C:\Users\Rony\Desktop\7mo semestre\TeoriaInformacion\AES>

RSA - BCrypt: práctica

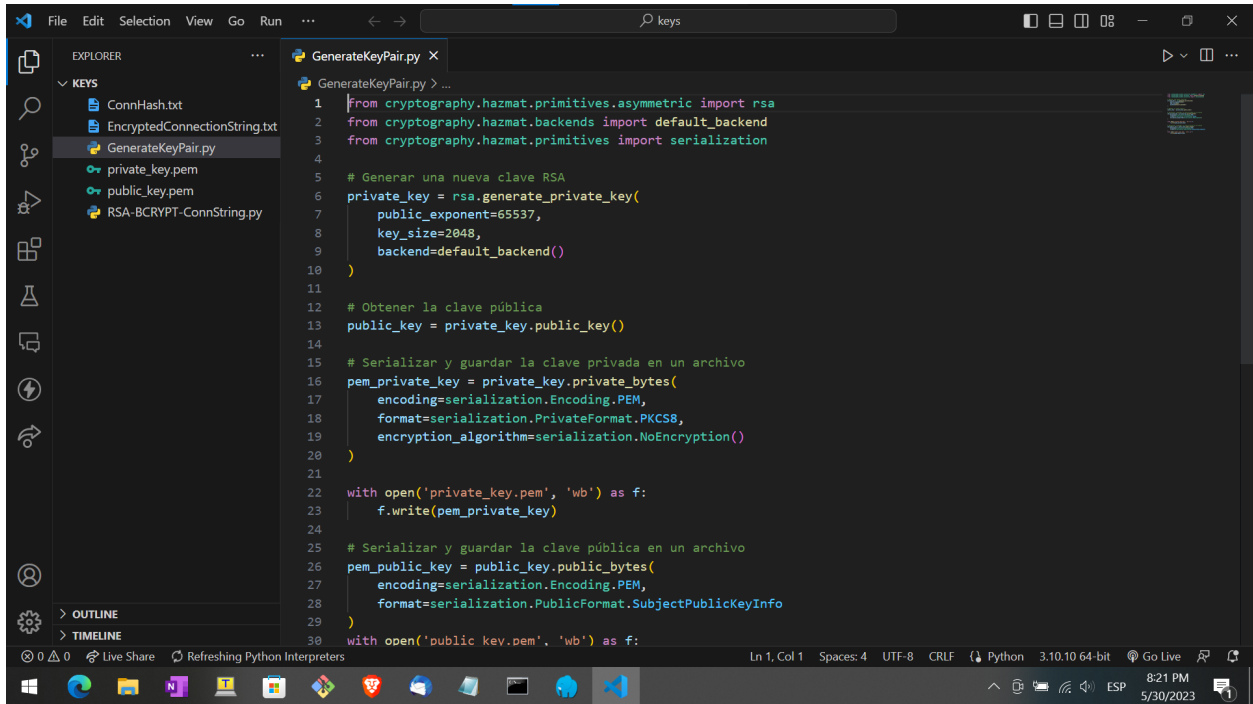
Cifrado con RSA

Para la práctica se ha realizado una App agenda, la cual está enfocada al almacenamiento y administración (creación, edición y eliminación) en una base de

datos de los contactos telefónicos, empresariales o personales de un usuario único en una computadora. El front-end de la aplicación fué desarrollado con React JS, y el back-end con Flask Python.

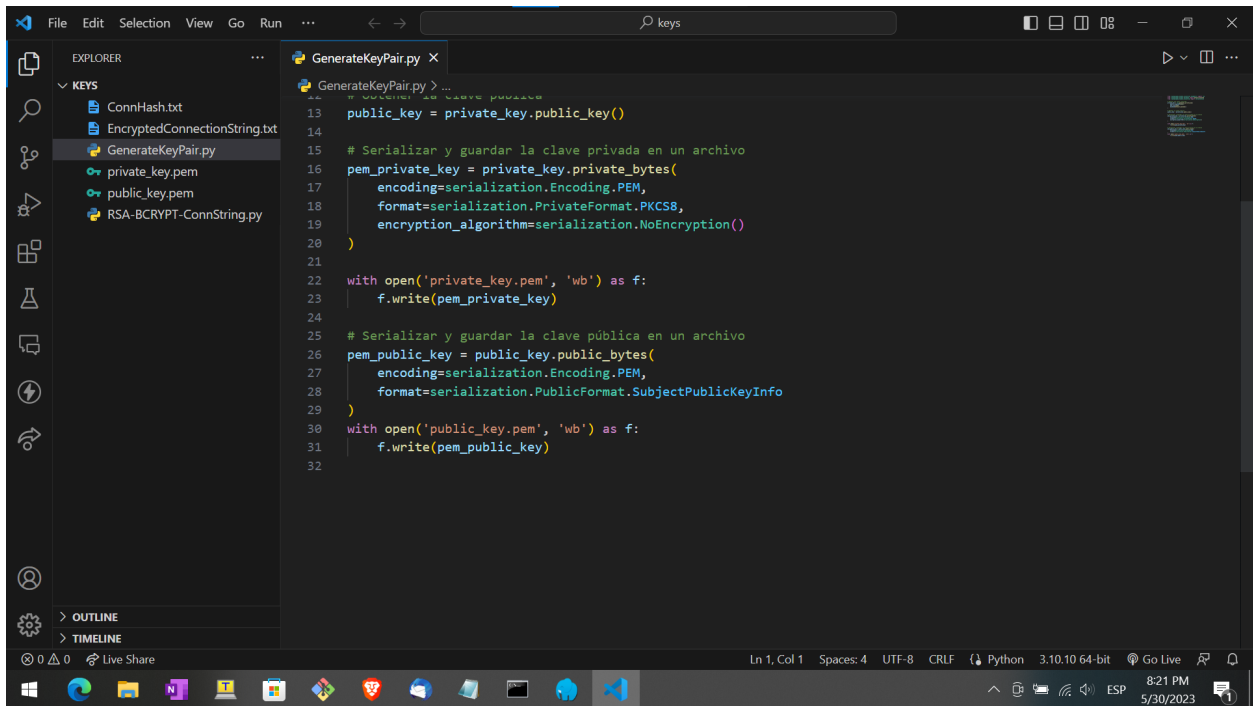


Para realizar el cifrado de la cadena de conexión se ha realizado un script de python que genera un par de claves de RSA, que seguidamente se almacenan en una ubicación segura de la computadora, a la que solamente el usuario con permisos en el ordenador puede acceder.



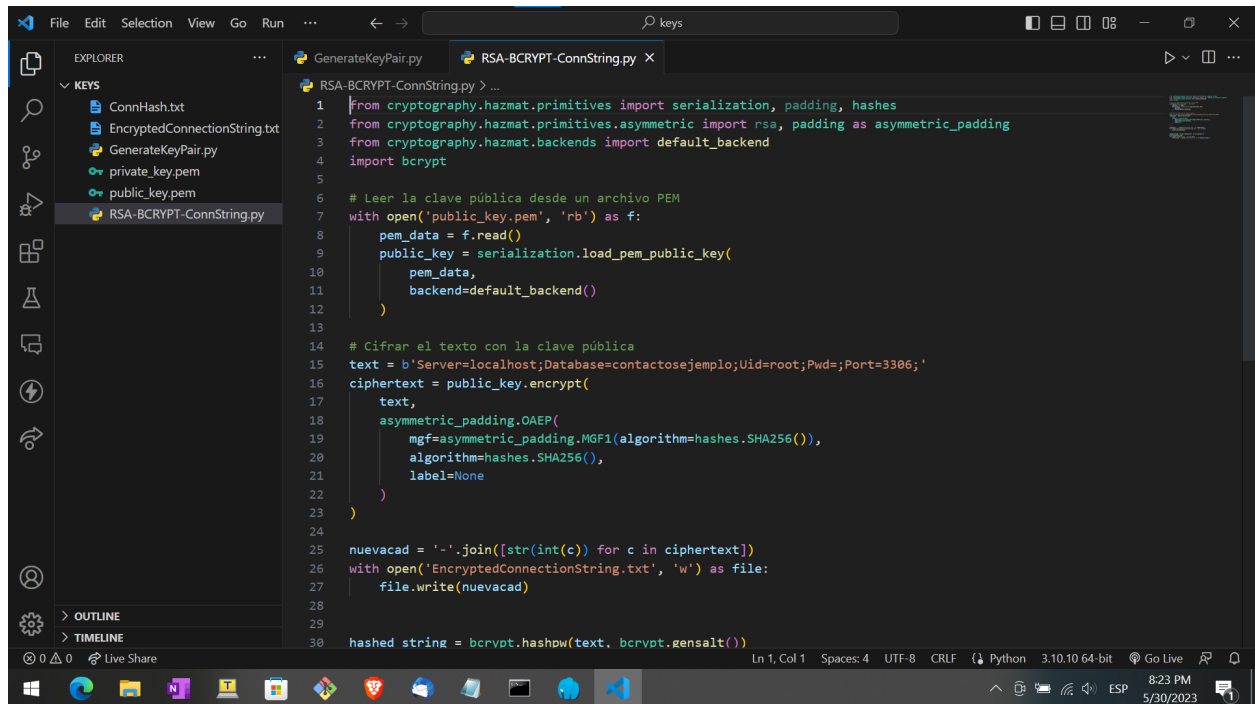
The screenshot shows the Visual Studio Code editor with the file explorer on the left displaying a project named 'KEYS'. The files listed are ConnHash.txt, EncryptedConnectionString.txt, GenerateKeyPair.py, private_key.pem, public_key.pem, and RSA-BCRYPT-ConnString.py. The main editor window shows the GenerateKeyPair.py script, which is a Python file using the cryptography library to generate RSA keys. The script includes comments in Spanish and code to generate a private key, serialize it to a PEM file, and then generate a public key and serialize it to another PEM file. The status bar at the bottom indicates the file is at line 1, column 1, with 4 spaces, UTF-8 encoding, and CRLF line endings. The system tray shows the time as 8:21 PM on 5/30/2023.

```
1 from cryptography.hazmat.primitives.asymmetric import rsa
2 from cryptography.hazmat.backends import default_backend
3 from cryptography.hazmat.primitives import serialization
4
5 # Generar una nueva clave RSA
6 private_key = rsa.generate_private_key(
7     public_exponent=65537,
8     key_size=2048,
9     backend=default_backend()
10 )
11
12 # Obtener la clave pública
13 public_key = private_key.public_key()
14
15 # Serializar y guardar la clave privada en un archivo
16 pem_private_key = private_key.private_bytes(
17     encoding=serialization.Encoding.PEM,
18     format=serialization.PrivateFormat.PKCS8,
19     encryption_algorithm=serialization.NoEncryption()
20 )
21
22 with open('private_key.pem', 'wb') as f:
23     f.write(pem_private_key)
24
25 # Serializar y guardar la clave pública en un archivo
26 pem_public_key = public_key.public_bytes(
27     encoding=serialization.Encoding.PEM,
28     format=serialization.PublicFormat.SubjectPublicKeyInfo
29 )
30
31 with open('public_key.pem', 'wb') as f:
```



This screenshot is identical to the one above, showing the same Visual Studio Code editor window with the GenerateKeyPair.py script. The file explorer on the left is the same, and the code in the main editor is the same. The status bar and system tray information are also identical.

Luego estas claves son leídas por otro script que cifra los datos de la cadena de conexión con la llave pública generada y guarda la cadena resultante como una serie de números, los cuales serán guardados de forma manual en el archivo .env de la aplicación del lado del backend.



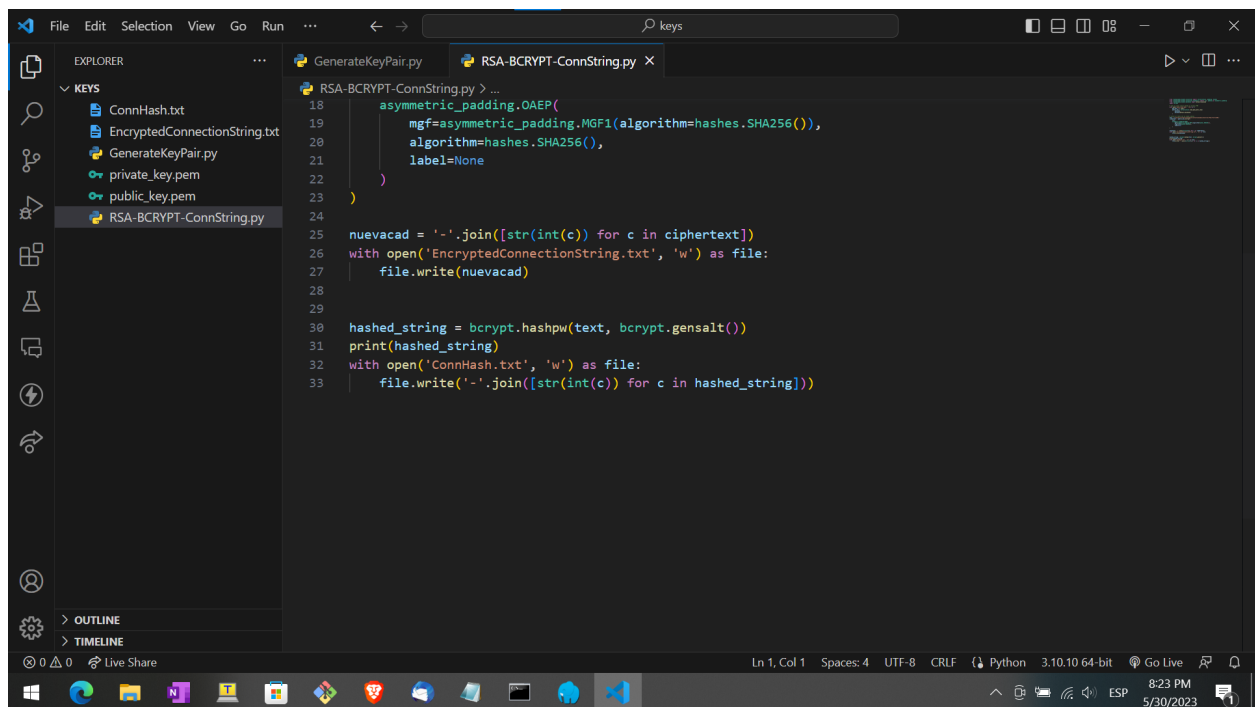
The screenshot shows the Visual Studio Code editor with the file `RSA-BCRYPT-ConnString.py` open. The Explorer sidebar on the left shows a project structure with files like `ConnHash.txt`, `EncryptedConnectionString.txt`, `GenerateKeyPair.py`, `private_key.pem`, `public_key.pem`, and `RSA-BCRYPT-ConnString.py`. The main editor displays the following Python code:

```
1 from cryptography.hazmat.primitives import serialization, padding, hashes
2 from cryptography.hazmat.primitives.asymmetric import rsa, padding as asymmetric_padding
3 from cryptography.hazmat.backends import default_backend
4 import bcrypt
5
6 # Leer la clave pública desde un archivo PEM
7 with open('public_key.pem', 'rb') as f:
8     pem_data = f.read()
9     public_key = serialization.load_pem_public_key(
10         pem_data,
11         backend=default_backend()
12     )
13
14 # Cifrar el texto con la clave pública
15 text = b'Server=localhost;Database=contactosejemplo;Uid=root;Pwd=;Port=3306;'
16 ciphertext = public_key.encrypt(
17     text,
18     asymmetric_padding.OAEP(
19         mgf=asymmetric_padding.MGF1(algorithm=hashes.SHA256()),
20         algorithm=hashes.SHA256(),
21         label=None
22     )
23 )
24
25 nuevacad = ''.join([str(int(c)) for c in ciphertext])
26 with open('EncryptedConnectionString.txt', 'w') as file:
27     file.write(nuevacad)
28
29
30 hashed_string = bcrypt.hashpw(text, bcrypt.gensalt())
```

The status bar at the bottom indicates the file is at line 1, column 1, using 4 spaces, UTF-8 encoding, CRLF line endings, and is a Python 3.10.10 64-bit file. The system clock shows 8:23 PM on 5/30/2023.

Hash con BCrypt

Seguidamente se ha implementado el guardado del hash de la cadena de conexión sin cifrar con bcrypt para su posterior verificación.



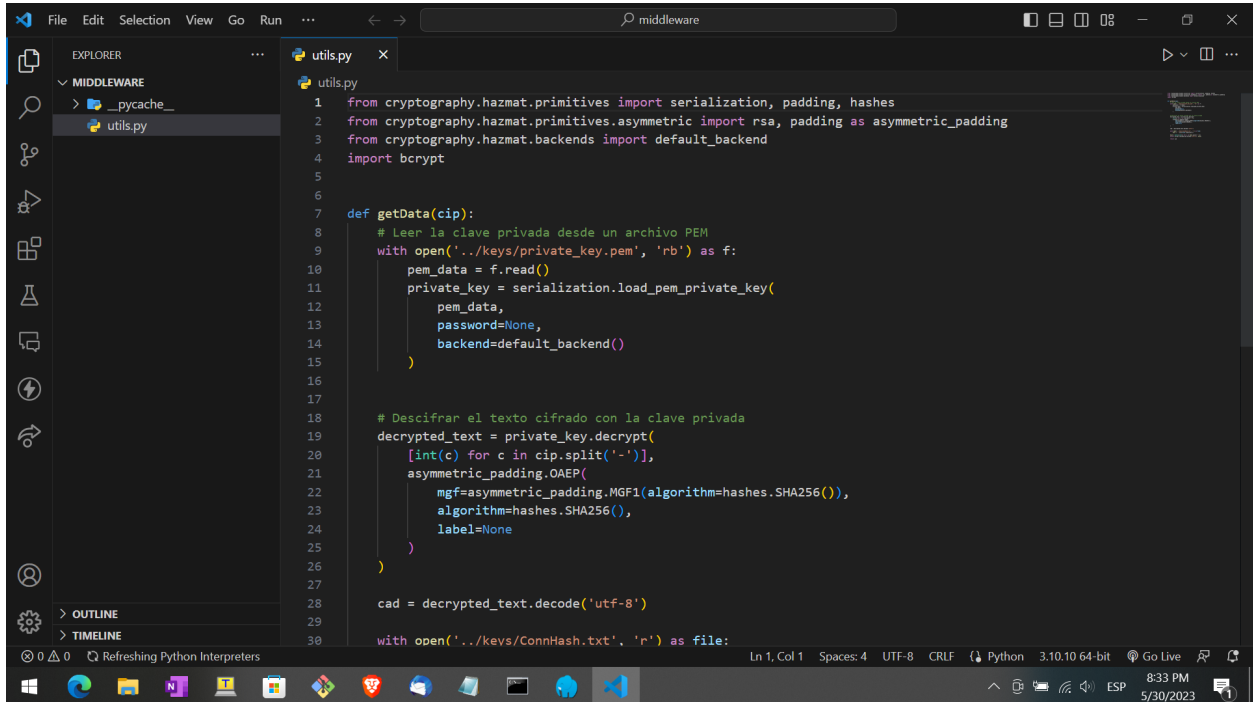
This screenshot shows the same Visual Studio Code editor with the `RSA-BCRYPT-ConnString.py` file. The code now includes a hash calculation and storage step. The relevant lines are:

```
18 asymmetric_padding.OAEP(
19     mgf=asymmetric_padding.MGF1(algorithm=hashes.SHA256()),
20     algorithm=hashes.SHA256(),
21     label=None
22 )
23 )
24
25 nuevacad = ''.join([str(int(c)) for c in ciphertext])
26 with open('EncryptedConnectionString.txt', 'w') as file:
27     file.write(nuevacad)
28
29
30 hashed_string = bcrypt.hashpw(text, bcrypt.gensalt())
31 print(hashed_string)
32 with open('ConnHash.txt', 'w') as file:
33     file.write(''.join([str(int(c)) for c in hashed_string]))
```

The status bar at the bottom remains the same, showing the file is at line 1, column 1, using 4 spaces, UTF-8 encoding, CRLF line endings, and is a Python 3.10.10 64-bit file. The system clock shows 8:23 PM on 5/30/2023.

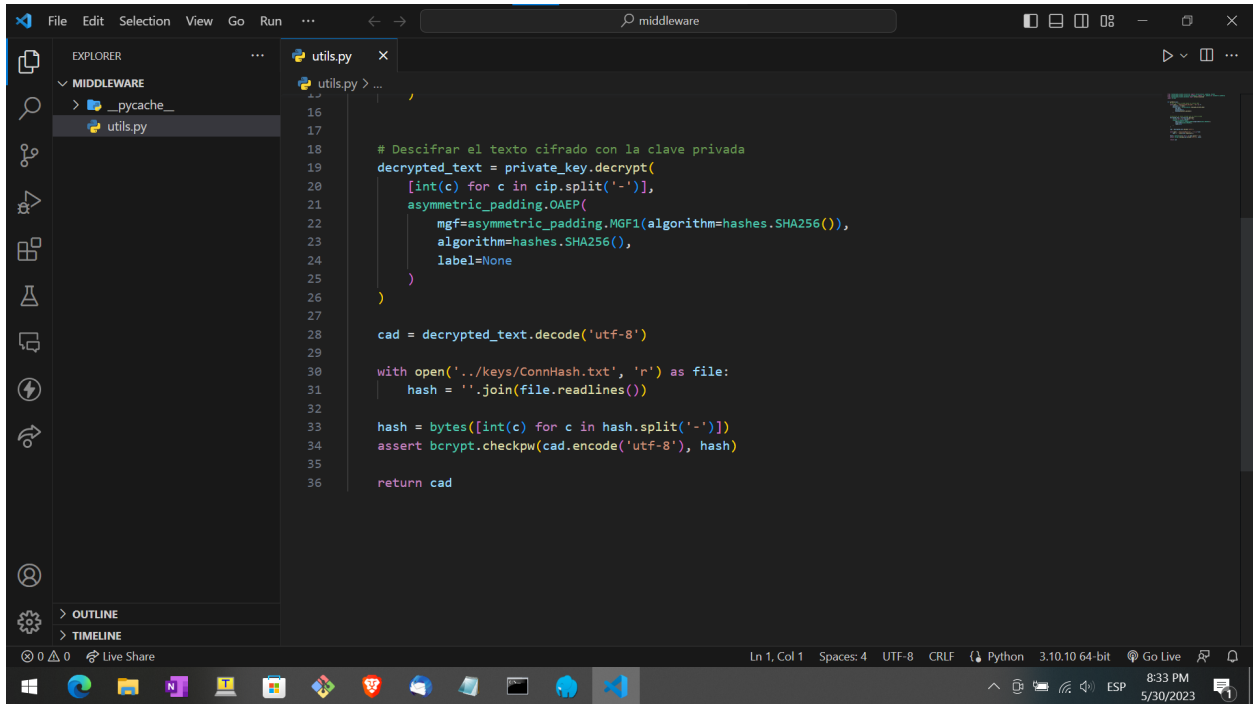
Autenticación a la base de datos

Se ha realizado un script de python que contiene una función la cual, de forma segura, recupera los datos de autenticación del gestor de la base de datos contenidos en la cadena de conexión cifrada. Esto se realiza al leer la llave privada de RSA para descifrar los datos, y luego se verifica la integridad al comparar el hash BCrypt de la cadena descifrada con el hash almacenado con anterioridad (el hash se encuentra almacenado en una ubicación segura de la computadora).



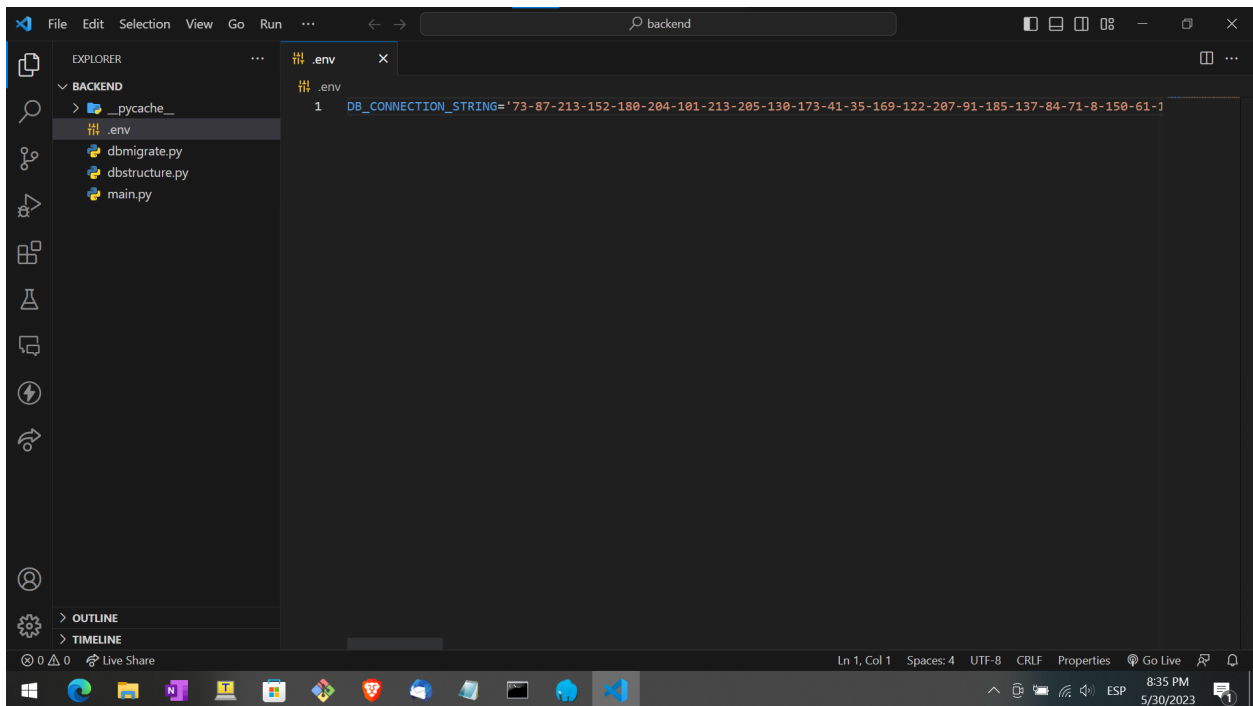
The image shows a screenshot of a Visual Studio Code editor window. The Explorer pane on the left shows a project structure with a folder named 'MIDDLEWARE' containing a file named 'utils.py'. The main editor area displays the contents of 'utils.py', which is a Python script. The script imports modules from the 'cryptography' library and defines a function 'getData(cip)' that reads a private RSA key from a file, decrypts a ciphertext 'cip', and then reads a hash from another file. The status bar at the bottom indicates the file is at line 1, column 1, with 4 spaces, in UTF-8 encoding, and the Python interpreter is set to 3.10.10 64-bit.

```
1 from cryptography.hazmat.primitives import serialization, padding, hashes
2 from cryptography.hazmat.primitives.asymmetric import rsa, padding as asymmetric_padding
3 from cryptography.hazmat.backends import default_backend
4 import bcrypt
5
6
7 def getData(cip):
8     # Leer la clave privada desde un archivo PEM
9     with open('../keys/private_key.pem', 'rb') as f:
10         pem_data = f.read()
11         private_key = serialization.load_pem_private_key(
12             pem_data,
13             password=None,
14             backend=default_backend()
15         )
16
17
18     # Descifrar el texto cifrado con la clave privada
19     decrypted_text = private_key.decrypt(
20         [int(c) for c in cip.split('-')],
21         asymmetric_padding.OAEP(
22             mgf=asymmetric_padding.MGF1(algorithm=hashes.SHA256()),
23             algorithm=hashes.SHA256(),
24             label=None
25         )
26     )
27
28     cad = decrypted_text.decode('utf-8')
29
30     with open('../keys/ConnHash.txt', 'r') as file:
```

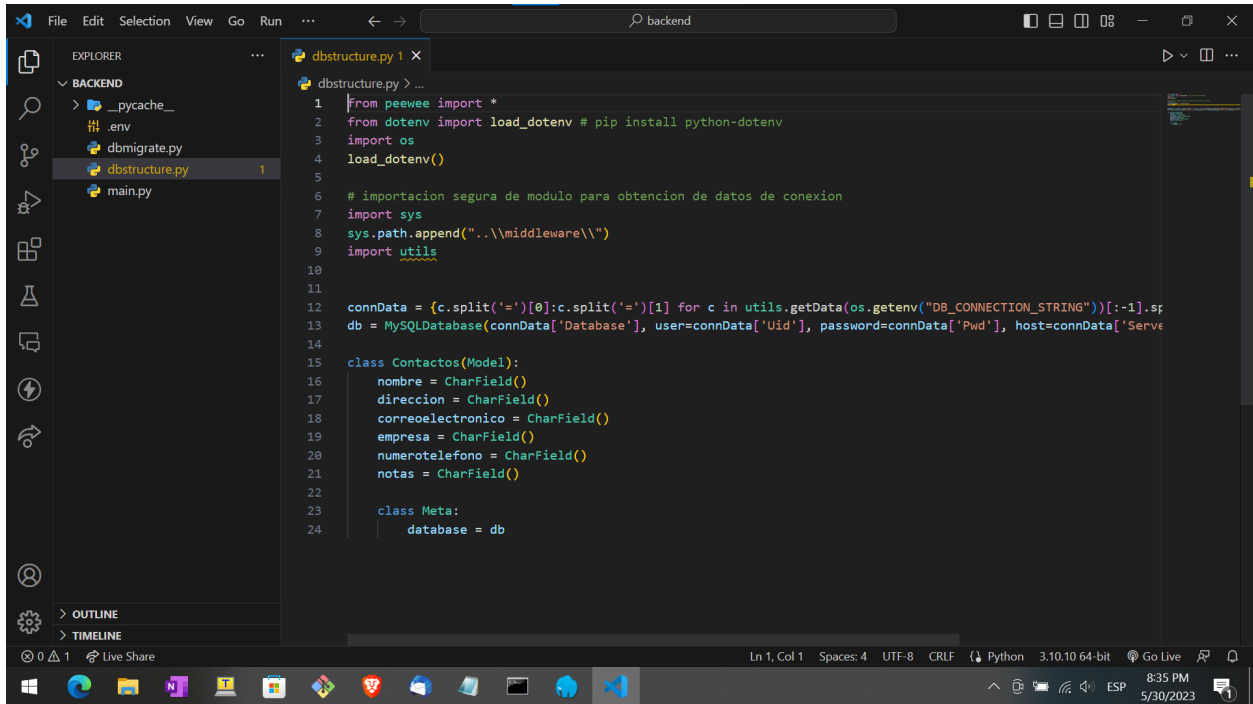


```
16
17
18 # Descifrar el texto cifrado con la clave privada
19 decrypted_text = private_key.decrypt(
20     [int(c) for c in cip.split('-')],
21     asymmetric_padding.OAEP(
22         mgf=asymmetric_padding.MGF1(algorithm=hashes.SHA256()),
23         algorithm=hashes.SHA256(),
24         label=None
25     )
26 )
27
28 cad = decrypted_text.decode('utf-8')
29
30 with open('../keys/ConnHash.txt', 'r') as file:
31     hash = ''.join(file.readlines())
32
33 hash = bytes([int(c) for c in hash.split('-')])
34 assert bcrypt.checkpw(cad.encode('utf-8'), hash)
35
36 return cad
```

La función recibe la cadena de conexión cifrada y devuelve los datos necesarios para la autenticación. En el back-end, esta función se llama al inicio de la ejecución del programa y se le envía la cadena contenida en el archivo .env bajo la propiedad DB_CONNECTION_STRING.

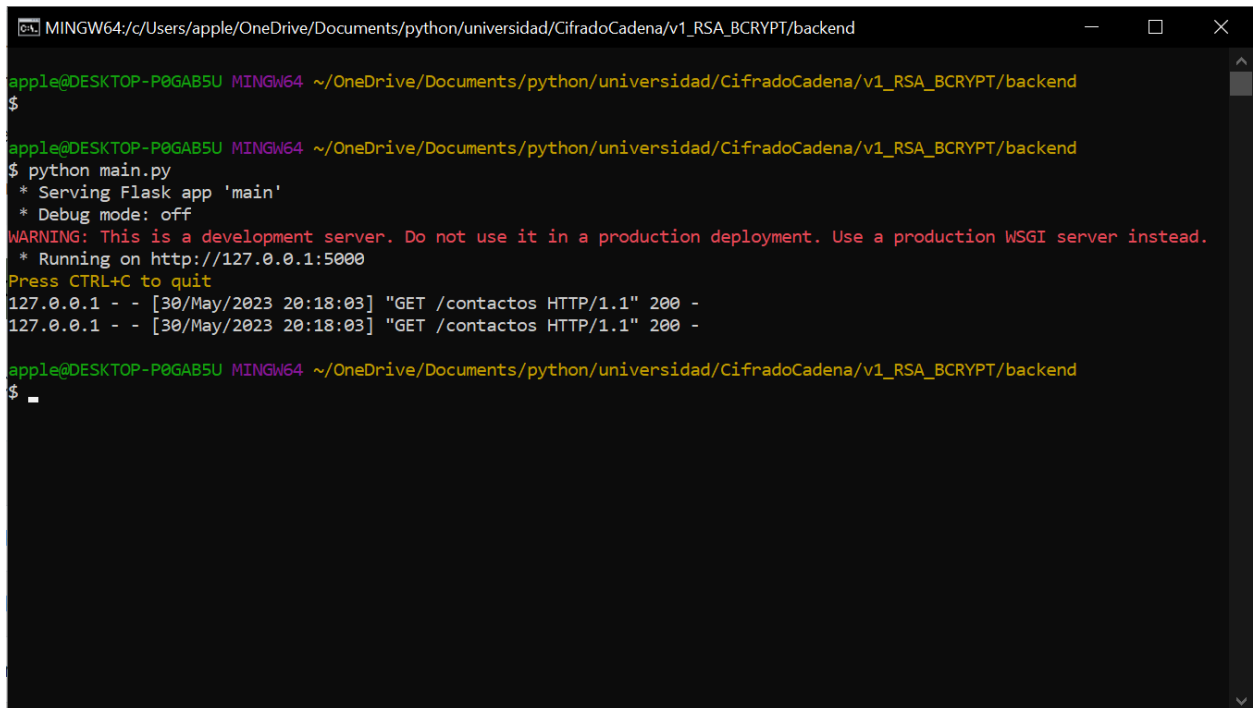


```
1 DB_CONNECTION_STRING='73-87-213-152-180-204-101-213-205-130-173-41-35-169-122-207-91-185-137-84-71-8-150-61-1'
```



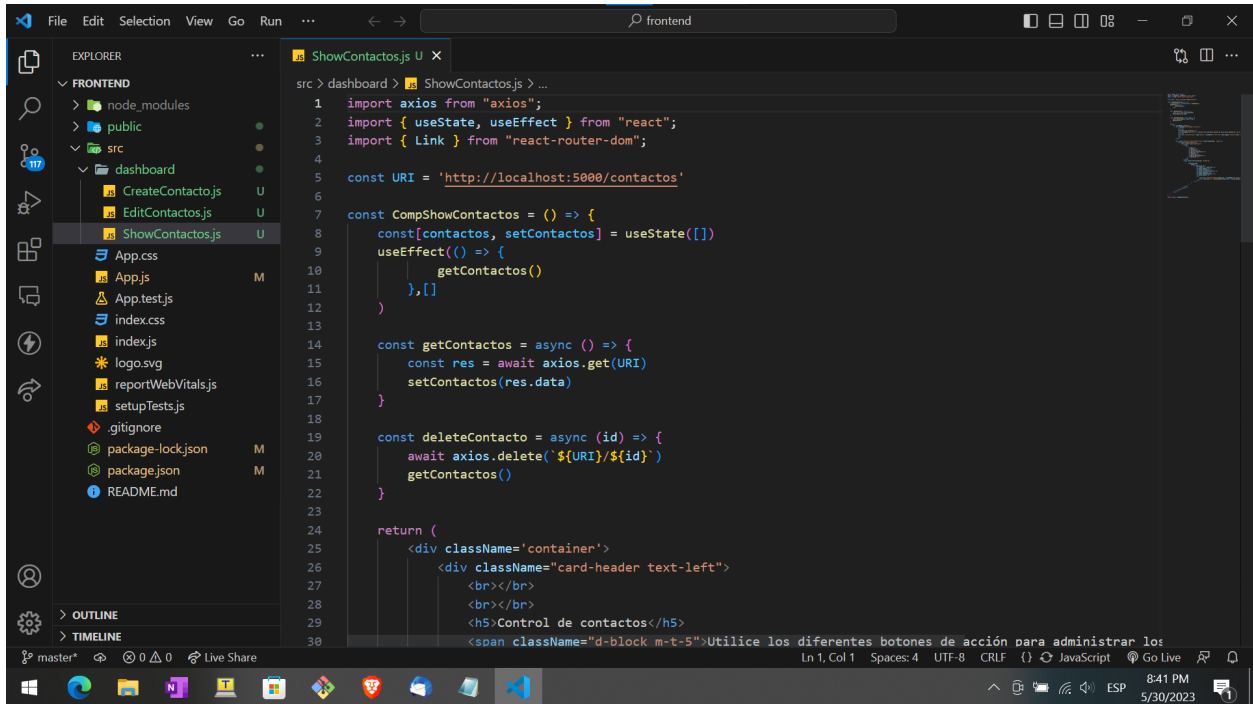
```
1 from peewee import *
2 from dotenv import load_dotenv # pip install python-dotenv
3 import os
4 load_dotenv()
5
6 # importacion segura de modulo para obtencion de datos de conexion
7 import sys
8 sys.path.append("../middleware\\")
9 import utils
10
11
12 connData = {c.split('=')[0]:c.split('=')[1] for c in utils.getData(os.getenv("DB_CONNECTION_STRING"))[:-1].split(';')}
13 db = MySQLDatabase(connData['Database'], user=connData['Uid'], password=connData['Pwd'], host=connData['Server'])
14
15 class Contactos(Model):
16     nombre = CharField()
17     direccion = CharField()
18     correoelectronico = CharField()
19     empresa = CharField()
20     numerotelefono = CharField()
21     notas = CharField()
22
23     class Meta:
24         database = db
```

Si el descifrado falla, o si el hash de BCrypt no coincide entonces se impide el inicio y se detiene la ejecución.



```
MINGW64/c:/Users/apple/OneDrive/Documents/python/universidad/CifradoCadena/v1_RSA_BCRYPT/backend
apple@DESKTOP-P0GAB5U MINGW64 ~/OneDrive/Documents/python/universidad/CifradoCadena/v1_RSA_BCRYPT/backend
$
apple@DESKTOP-P0GAB5U MINGW64 ~/OneDrive/Documents/python/universidad/CifradoCadena/v1_RSA_BCRYPT/backend
$ python main.py
* Serving Flask app 'main'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [30/May/2023 20:18:03] "GET /contactos HTTP/1.1" 200 -
127.0.0.1 - - [30/May/2023 20:18:03] "GET /contactos HTTP/1.1" 200 -
apple@DESKTOP-P0GAB5U MINGW64 ~/OneDrive/Documents/python/universidad/CifradoCadena/v1_RSA_BCRYPT/backend
$
```

Desde el front-end se realizan peticiones al back-end para recuperar la información.



```
1 import axios from "axios";
2 import { useState, useEffect } from "react";
3 import { Link } from "react-router-dom";
4
5 const URI = 'http://localhost:5000/contactos'
6
7 const CompShowContactos = () => {
8   const [contactos, setContactos] = useState([])
9   useEffect(() => {
10     getContactos()
11   }, [])
12
13   const getContactos = async () => {
14     const res = await axios.get(URI)
15     setContactos(res.data)
16   }
17
18   const deleteContacto = async (id) => {
19     await axios.delete(`${URI}/${id}`)
20     getContactos()
21   }
22
23   return (
24     <div className='container'>
25       <div className="card-header text-left">
26         <br></br>
27         <br></br>
28         <h5>Control de contactos</h5>
29         <span className="d-block m-t-5">Utilice los diferentes botones de acción para administrar los
30
```