más allá de esta hora, el avión se considera en retraso.

Universidad Carlos III

Curso Criptografía y Seguridad Informática 2024-25

Práctica I

Curso 2024-25

Criptografía

Grado Ing Informática

Fecha: **30/10/2024** - ENTREGA: **I**

GRUPO: **80***

Alumnos:

Hector Álvarez Marcos 100495794

Jorge Agramunt Muro 100495764

Índice:

Grupo: 80

ID de grupo de prácticas: 2

Nombre de todos los alumnos: Jorge Agramunt Muro y Héctor Álvarez Marcos

Correo de todos los alumnos: 100495764@alumnos.uc3m.es y 100495794@alumnos.uc3m.es

Repositorio de código: https://github.com/CodificandoAlonso/P1_CRIPTO

1. Propósito y Estructura de la Aplicación

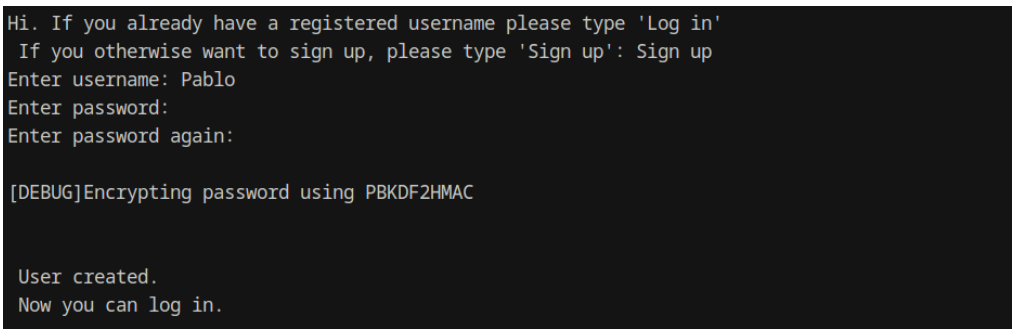
Propósito
La aplicación es un sistema de mensajería y gestión de productos que permite a los usuarios registrarse, iniciar sesión, agregar productos para la venta, comprar productos y enviar mensajes cifrados entre usuarios.

Estructura Interna
La aplicación está dividida en varios módulos:

- main.py: Punto de entrada de la aplicación. Maneja la interacción del usuario y la lógica principal.
- user.py: Maneja la creación y validación de usuarios, así como la generación de claves.
- server.py: Gestiona la lógica del servidor, incluyendo la verificación de usuarios, la gestión de productos y el cifrado/descifrado de datos.
- messages.py: Gestiona el envío y recepción de mensajes cifrados entre usuarios.

2. Autenticación de usuarios

Para la autenticación de usuarios, se utiliza el algoritmo PBKDF2 (Password-Based Key Derivation Function 2) con HMAC-SHA256. Este algoritmo es robusto y ampliamente utilizado para derivar claves criptográficas a partir de contraseñas. Es el único caso en el que generamos claves a partir de contraseñas



```python
def save_user(self, username, password):
    print("\n[DEBUG]Encrypting password using PBKDF2HMAC\n")
    users = self.access_server.open_and_return_jsons('jsones/users.json')
    password = password.encode("utf-8")
    salt = os.urandom(16)
    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        length=32,
        salt=salt,
        iterations=480000,
    )
    key = base64.urlsafe_b64encode(kdf.derive(password))
    users.append({'username': username, 'token': str(key), 'id':str(salt)})
    self.access_server.save_jsons(users,'jsones/users.json')
```

Validación: Las contraseñas deben tener al menos 8 caracteres, incluir una letra mayúscula, un dígito y un carácter especial.
Derivación de Claves: Se utiliza PBKDF2HMAC para derivar una clave a partir de la contraseña del usuario. Esta clave se almacena junto con una sal (salt) generada aleatoriamente.
Almacenamiento: Las contraseñas derivadas y las sales se almacenan en un archivo JSON cifrado.

3. Cifrado simétrico

Algoritmos Utilizados
Se utiliza tanto para la encriptación simétrica de la BBDD como para la generación de claves simétricas a la hora de intercambiar mensajes con diferentes usuarios.

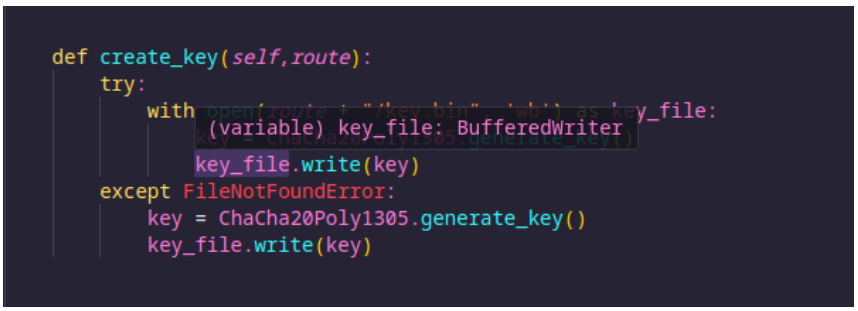
Se utiliza el algoritmo ChaCha20Poly1305 para el cifrado simétrico. Este algoritmo es rápido y seguro, y proporciona cifrado autenticado, lo que garantiza la integridad y confidencialidad de los datos.

Gestión de Claves
Generación: La clave simétricas para la base de datos se generan utilizando ChaCha20Poly1305.generate_key().
Almacenamiento: Las claves se almacenan en archivos binarios cifrados.
Uso: Las claves se utilizan para cifrar y descifrar los archivos JSON que contienen los datos de usuarios, productos y mensajes.

```
Welcome to the app, if anytime you want to exit, press Ctrl + C.
 While you are logged in, if you press Ctrl + E, you will be logged out
 "[DEBUG] THIS DATABASES AND THE DATA ARE BEING PROTECTED WITH ChaCha20Poly1305
```

```python
class Server():
    def __init__(self):
        self.message = Message(self)
        server_dir = os.path.join('keys', "Server")
        """Primera instanciacion del servidor. Creamos localizacion de clave
        para desencriptar los jsones y creamos los jsones encriptados"""
        if not os.path.isdir(server_dir):
            os.makedirs(server_dir, exist_ok=True)
            self.create_key(server_dir)
            self.__key = self.get_key(server_dir)
            self.create_jsones()
        self.__key = self.get_key(server_dir)
```

(parameter) self: Self@Server



Se usa el algoritmo de cifrado simétrico de Fernet para generar una clave simétrica por cada conversación entre 2 usuarios. Dicha clave se guarda en un json cifrada con la pública de los 2 interlocutores, añadiendo el originador de la conversación una firma que garantiza autenticación del envío de la clave.

4. Cifrado asimétrico

Algoritmos Utilizados
Se utiliza RSA con un tamaño de clave de 4096 bits para el cifrado asimétrico. Este algoritmo es seguro y adecuado para cifrar claves simétricas y mensajes cortos.

Gestión de Claves
Generación: Cada usuario tiene un par de claves RSA (pública y privada) generadas al momento de registrarse.
Almacenamiento: Las claves privadas se almacenan en archivos PEM sin cifrar, mientras que las claves públicas se almacenan en archivos PEM públicos.
Sabemos que la clave privada no debería de guardarse en la base de datos, pero puesto que nuestra aplicación no permite el uso simultáneo multicuenta, para simular conversaciones y que se puedan acceder a los datos asumimos que esos datos no deberían ser guardados.
Uso: Las claves públicas se utilizan para cifrar las claves simétricas antes de enviarlas a otros usuarios. Las claves privadas se utilizan para descifrar las claves simétricas recibidas.
También usamos la pública de cada usuario para guardar un registro de los mensajes leídos, ya que si se guardara con la simétrica de la conversación, como esta se elimina al cerrar la aplicación no se podría acceder a su contenido, por lo que cifrando cada usuario estos mensajes con la privada del propio usuario(Como si quisiera enviar un mensaje a sí mismo), garantizamos que solo él pueda acceder al contenido.
Códigos de Autenticación de Mensajes:
Nuestro modo de autenticación de mensajes se explica anteriormente pero básicamente se compone de una clave simétrica entre interlocutores que se guarda con las claves públicas de los usuarios, para garantizar que solo ellos sean capaces de descifrar el contenido de la clave simétrica. Para darle autenticación, se envía una firma hecha con la clave privada del creador de la conversación. Esto garantiza que el contenido de la clave simétrica cifrada no ha sido alterado.