Estudiantes: Jesús David Ramírez Celis **Código: 2211593**

Juan Diego Roa Porras **Código: 2210086**

Laboratorio Seguridad en los sistemas de información

Laboratorio 1: Cifrado de Mensajes con Python

Introducción al Proyecto

En este laboratorio vamos a construir un sistema sencillo pero robusto que nos permitirá cifrar y descifrar mensajes usando Python junto con la librería cryptography. Para hacerlo más realista, utilizaremos RabbitMQ como intermediario. La idea es simular que varios usuarios están intercambiando mensajes cifrados y asegurarnos de que únicamente quien tenga la clave correcta pueda leerlos. Para poner a prueba el sistema, incluiremos errores intencionales y así validar que detecte cualquier alteración.

Algunos Conceptos Claves

¿Qué es el Cifrado Simétrico?

El cifrado simétrico consiste en utilizar una sola clave secreta tanto para cifrar como para descifrar mensajes. Es fundamental que esta clave permanezca segura y que solo las partes involucradas en la comunicación tengan acceso a ella. Fernet es un método específico que ofrece confidencialidad e integridad y es implementado por la librería cryptography.

¿Qué es la Integridad?

La integridad busca asegurar que la información que se recibe sea exactamente la misma que fue enviada. El método Fernet que usaremos permite detectar si el mensaje fue alterado, garantizando que el mensaje recibido es exactamente igual al original.

Tecnologías y Herramientas

- Python 3.10.16
- cryptography (Fernet) para cifrado simétrico
- RabbitMQ para gestionar los mensajes
- FastAPI y Uvicorn para crear la API backend
- Docker para desplegar fácilmente los servicios

Instalación y Preparativos

Para arrancar nuestro proyecto, primero instalamos todas las herramientas necesarias ejecutando:

```
pip install fastapi uvicorn pika pydantic cryptography
```

Desarrollo Paso a Paso

Paso 1: Creación de la Clave Simétrica

Usaremos Fernet para generar nuestra clave segura:

```
from cryptography.fernet import Fernet
key = Fernet.generate_key()
print(key.decode())
```

Debemos almacenar esta clave en una variable de entorno para su posterior uso:

```
SECRET_KEY = os.getenv("SECRET_KEY",
"AhAmNzQqSBL67zbvlXPAflI5NV5lJD6rxUjbuTC00-s=")
```

Paso 2: Cifrado del Mensaje

Con Fernet ciframos el mensaje antes de enviarlo:

```
mensaje_json = message.model_dump_json()
mensaje_bytes = mensaje_json.encode('utf-8')
mensaje_cifrado = fernet.encrypt(mensaje_bytes)
```

Luego enviamos este mensaje cifrado al RabbitMQ usando la ruta /publishMessage.

Paso 3: Descifrado del Mensaje

El servidor backend recibe estos mensajes cifrados desde RabbitMQ y los descifra con la clave simétrica:

```
mensaje_descifrado = fernet.decrypt(mensaje_cifrado)
```

Paso 4: Simulando Errores de Integridad

Para demostrar cómo detectamos alteraciones en el cifrado, introducimos un 20% de probabilidad de usar una clave incorrecta al intentar descifrar el mensaje, generando así un error de integridad:

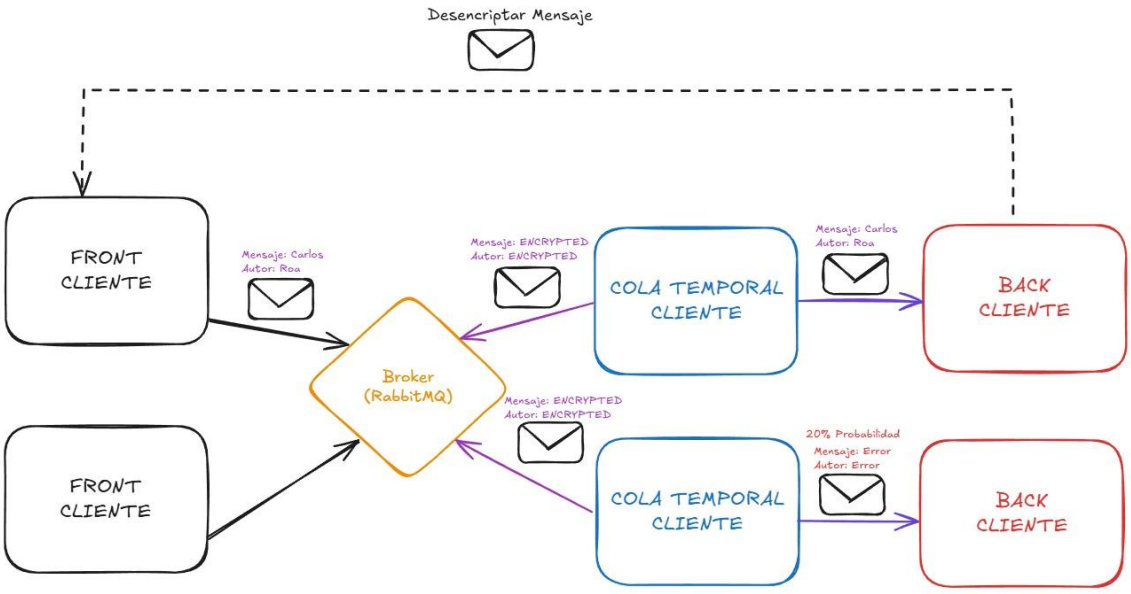
```
if random.random() > 0.8:
    mensaje_descifrado = fernet_falso.decrypt(mensaje_cifrado)
```

```
else:
    mensaje_descifrado = fernet.decrypt(mensaje_cifrado)
```

Esto nos ayuda a entender claramente cómo el cifrado detecta cuando se ha manipulado un mensaje.

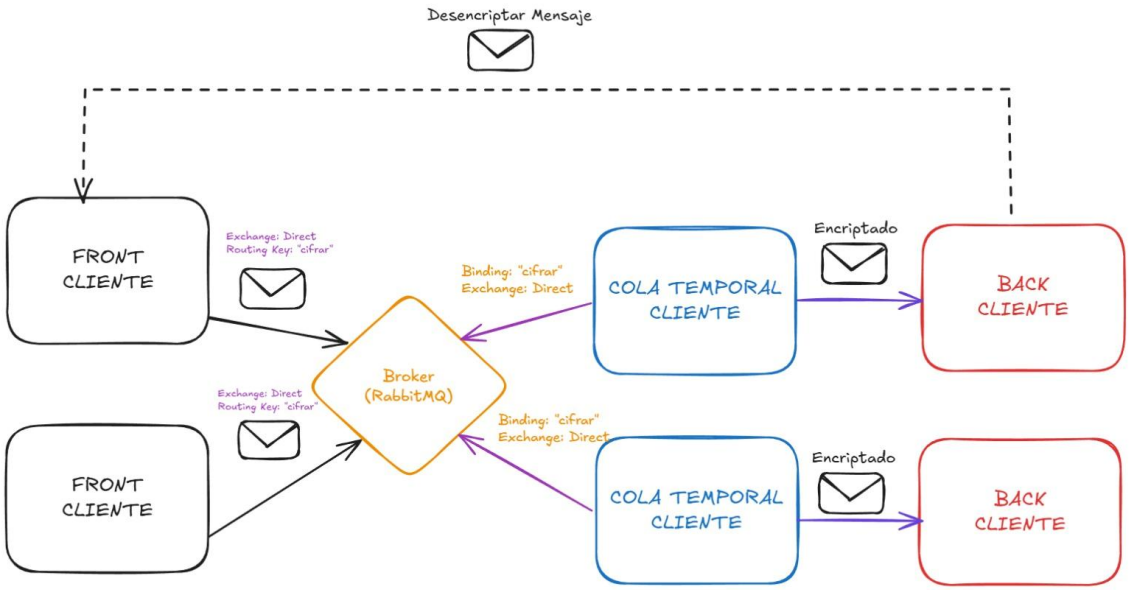
Explicación visual del proceso:

Flujo básico:



- El usuario envía un mensaje cifrado al RabbitMQ.
- RabbitMQ lo enruta hacia una cola temporal.
- El servidor lo descifra y procesa.

Flujo con seguridad (Cifrado e integridad):



- Siempre se cifran los mensajes antes del envío.
- Con frecuencia (20%), simulamos errores intencionales para probar cómo reaccionaría el sistema ante un ataque real.

Desplegado con Docker:

```
docker compose up -d
```

Conclusión:

Este laboratorio permitió desarrollar una comprensión profunda de la aplicación práctica de técnicas criptográficas, específicamente del cifrado simétrico. Se logró evidenciar claramente cómo esta técnica protege los datos sensibles y cómo el sistema reacciona eficazmente ante intentos de alteración o ataques simulados, fortaleciendo así la importancia de mantener la integridad y confidencialidad en las comunicaciones digitales. Además, se adquirieron habilidades técnicas valiosas mediante el uso de herramientas modernas como RabbitMQ y Docker, que serán fundamentales en futuros desarrollos profesionales en el área de seguridad informática.

Recursos complementarios:

URL del repositorio en Github: https://github.com/JuanRoa785/Cifrado-Mensajes-Python.git