Autenticação com Token e Mecanismos de Segurança em Sistemas de Mensageria.

Para criar um algoritmo de autenticação e segurança em um sistema de mensageria, é importante considerar alguns conceitos básicos de segurança de informação: Autenticação, autorização, confidencialidade, integridade e não-repúdio.

1. **Autenticação: Verificar a identidade do usuário.**
2. **Criptografia: Garantir a confidencialidade das mensagens.**
3. **Assinatura digital: Garantir a integridade e não-repúdio das mensagens.**

1. Autenticação

Usaremos uma combinação de nome de usuário e senha. A senha será armazenada de forma segura usando hash.

```
import hashlib
import hmac
# Função para criar o hash da senha
def hash_password(password) :
    return
hashlib.sha256(password.encode()).hexdigest()

# Dicionário para armazenar usuários e senhas
users = {
    "user1":
hash_password("password123"),
    "user2":
hash_password("mysecurepassword")
}

# Função para autenticar o usuário
def authenticate(username, password):
    if username in users and
hmac.compare_digest(users[username]),
hash_password(password)):
        return True
    else:
        return False

# Exemplo de uso
username = input("Username: ")
password = input("Password :")
```

```
if authenticate(username, password):
      print("Authenticated successfully")
else:
      print("Authentication failed")
```

2. Criptografia

Vamos usar a biblioteca cryptography para criptografar e descriptografar mensagens.

Primeiro, instale a biblioteca:

```
pip install cryptography
```

então, use o seguinte código:

```
from cryptography.fernet import Fernet

# Gerar uma chave para criptografia
key = Fernet.generate_key()
cipher_suite = Fernet(key)

# Função para criptografar uma mensagem
def encrypt_message(message):
    return
cipher_suite.encrypt(message.encode())

# Função para descriptografar uma mensagem
def decrypt_message(encrypted_message):
    return
cipher_suite.decrypt(encrypted_message).decode()

# Exemplo de uso
message = "This is secret message"
encrypted_message =
encrypt_message(message)
print("Encrypted:", encrypted_message)

decrypted_message =
decrypt_message(encrypted_message)
print("Decrypted:", decrypted_message)
```

3. Assinatura Digital

Vamos usar a biblioteca cryptography para criar e verificar assinaturas digitais.

```
from
cryptography.hazmat.primitives.asymmetric import rsa, padding
from cryptography.hazmat.primitives
```

```
import hashes, serialization

# Gerar chave privada
private_key = rsa.generate_private_key(public_exponent=65537,
key_size=2048)

# Extrair chave pública
public_key = private_key.public_key()

# Função para assinar uma mensagem
def sign_message(message):
    signature = private_key.sign(message.encode(),padding.PSS(

mgf=padding.MGF1(hashes.SHA256()),

salt_length=padding.PSS.MAX_LENGTH
    ),
    hashes.SHA256()

    )
    return signature

# Função para verificar a assinatura
def verify_signature(message,signature):
    try:
        public_key.verify(signature,message.encode(),padding.PSS(

mgf=padding.MGF1(hashes.SHA256()),

salt_lenght=padding.PSS.MAX_LENGHT
        ),
        hashes.SHA256()
        )
        return True
    except:
        return False

# Exemplo de uso
message = "This is a signed message"
signature = sign_message(message)
print("Signature:", signature)

if verify_signature(message, signature):
    print("Signature verified")
else:
    print("Signature verification failed")
```

Este sistema permitirá que usuários se registrem, façam login, enviem mensagens criptografadas e assinadas, e que os destinatários possam descriptografar e verificar a assinatura das mensagens recebidas

ESTRUTURA DO SISTEMA

1. **Registro e Autenticação de usuários**
2. **Criptografia de mensagens**
3. **Assinatura Digital de Mensagens**
4. **Envio e recebimento de Mensagens**

Requisitos

Python 3.6+

Bibliotecas

Cryptography

Pickle (para armazenar dados de forma simples)

```
pip install cryptography
```

implementação

vamos dividir o sistema em varias partes para facilitar a compreensão

1. **Gerenciamento de Usuários**

   Primeiro, precisamos de um sistema para registrar e autenticar usuários. Utilizaremos hashing para senhas e armazenaremos os usuários em um arquivo usando pickle.

```python
# user_management.py

import hashlib
import hmac
import pickle
import os

USER_DATA_FILE = 'users.pkl'

def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()
```

```
def load_users():
    if not os.path.exists(USER_DATA_FILE):
        return {}
    with open(USER_DATA_FILE, 'rb') as f:
        return pickle.load(f)

def save_users(users):
    with open(USER_DATA_FILE, 'wb') as f:
        pickle.dump(users, f)

def register(username, password):
    users = load_users()
    if username in users:
        print("Usuário já existe.")
        return False
    users[username] = hash_password(password)
    save_users(users)
    print("Usuário registrado com sucesso.")
    return True

def authenticate(username, password):
    users = load_users()
    if username in users and hmac.compare_digest(users[username],
hash_password(password)):
        print("Autenticado com sucesso.")
        return True
    else:
        print("Falha na autenticação.")
        return False
```

2. Criptografia e Assinatura Digital

Utilizaremos a biblioteca cryptography para criptografar e assinar mensagens. Cada usuário terá um par de chaves (privada e pública).

```
# crypto_utils.py

from cryptography.fernet import Fernet
from cryptography.hazmat.primitives.asymmetric import rsa, padding
from cryptography.hazmat.primitives import hashes, serialization
import os

KEY_DIR = 'keys'

def generate_keys(username):
    if not os.path.exists(KEY_DIR):
        os.makedirs(KEY_DIR)
```

```
    private_key_path = os.path.join(KEY_DIR,
f"{username}_private_key.pem")
    public_key_path = os.path.join(KEY_DIR, f"{username}_public_key.pem")

    if os.path.exists(private_key_path) and
os.path.exists(public_key_path):
        print("Chaves já existem.")
        return

    private_key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=2048
    )

    # Salvar chave privada
    with open(private_key_path, 'wb') as f:
        f.write(private_key.private_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PrivateFormat.TraditionalOpenSSL,
            encryption_algorithm=serialization.NoEncryption()
        ))

    # Salvar chave pública
    public_key = private_key.public_key()
    with open(public_key_path, 'wb') as f:
        f.write(public_key.public_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PublicFormat.SubjectPublicKeyInfo
        ))

    print("Chaves geradas e salvas.")

def load_private_key(username):
    private_key_path = os.path.join(KEY_DIR,
f"{username}_private_key.pem")
    with open(private_key_path, 'rb') as f:
        return serialization.load_pem_private_key(
            f.read(),
            password=None
        )

def load_public_key(username):
    public_key_path = os.path.join(KEY_DIR, f"{username}_public_key.pem")
    with open(public_key_path, 'rb') as f:
        return serialization.load_pem_public_key(
            f.read()
        )

def generate_symmetric_key():
```

```
    return Fernet.generate_key()

def load_symmetric_key(key_file):
    with open(key_file, 'rb') as f:
        return f.read()

def save_symmetric_key(key, key_file):
    with open(key_file, 'wb') as f:
        f.write(key)
```

3. Sistema de Mensageria

Este módulo permitirá que usuários enviem e recebam mensagens criptografadas e assinadas.

```
# messaging_system.py

import os
from user_management import register, authenticate
from crypto_utils import generate_keys, load_private_key,
load_public_key, generate_symmetric_key, save_symmetric_key
from cryptography.fernet import Fernet
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.primitives import hashes
import pickle

MESSAGES_FILE = 'messages.pkl'

def load_messages():
    if not os.path.exists(MESSAGES_FILE):
        return []
    with open(MESSAGES_FILE, 'rb') as f:
        return pickle.load(f)

def save_messages(messages):
    with open(MESSAGES_FILE, 'wb') as f:
        pickle.dump(messages, f)

def send_message(sender, recipient, message):
    # Carregar chave pública do destinatário
    try:
        recipient_public_key = load_public_key(recipient)
    except FileNotFoundError:
        print("Chave pública do destinatário não encontrada.")
        return False

    # Gerar chave simétrica para esta mensagem
    symmetric_key = generate_symmetric_key()
    cipher = Fernet(symmetric_key)
    encrypted_message = cipher.encrypt(message.encode())
```

```
    # Criptografar a chave simétrica com a chave pública do destinatário
    encrypted_symmetric_key = recipient_public_key.encrypt(
        symmetric_key,
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
    )

    # Assinar a mensagem
    sender_private_key = load_private_key(sender)
    signature = sender_private_key.sign(
        encrypted_message,
        padding.PSS(
            mgf=padding.MGF1(hashes.SHA256()),
            salt_length=padding.PSS.MAX_LENGTH
        ),
        hashes.SHA256()
    )

    # Salvar a mensagem
    messages = load_messages()
    messages.append({
        'sender': sender,
        'recipient': recipient,
        'encrypted_symmetric_key': encrypted_symmetric_key,
        'encrypted_message': encrypted_message,
        'signature': signature
    })
    save_messages(messages)

    print("Mensagem enviada com sucesso.")
    return True

def receive_messages(username):
    messages = load_messages()
    user_messages = [msg for msg in messages if msg['recipient'] ==
username]

    if not user_messages:
        print("Nenhuma mensagem para você.")
        return

    # Carregar chave privada do usuário
    private_key = load_private_key(username)

    for idx, msg in enumerate(user_messages, 1):
```

```
        print(f"\nMensagem {idx}:")
        print(f"De: {msg['sender']}")

        # Descriptografar a chave simétrica
        symmetric_key = private_key.decrypt(
            msg['encrypted_symmetric_key'],
            padding.OAEP(
                mgf=padding.MGF1(algorithm=hashes.SHA256()),
                algorithm=hashes.SHA256(),
                label=None
            )
        )
        cipher = Fernet(symmetric_key)
        decrypted_message =
cipher.decrypt(msg['encrypted_message']).decode()
        print(f"Mensagem: {decrypted_message}")

        # Verificar a assinatura
        sender_public_key = load_public_key(msg['sender'])
        try:
            sender_public_key.verify(
                msg['signature'],
                msg['encrypted_message'],
                padding.PSS(
                    mgf=padding.MGF1(hashes.SHA256()),
                    salt_length=padding.PSS.MAX_LENGTH
                ),
                hashes.SHA256()
            )
            print("Assinatura: Verificada com sucesso.")
        except:
            print("Assinatura: Falha na verificação.")

    # Opcional: remover as mensagens após leitura
    # messages = [msg for msg in messages if msg['recipient'] !=
username]
    # save_messages(messages)
```

4. Interface do Usuário

Finalmente, criaremos uma interface simples no terminal para interagir com o sistema.

```
# main.py

from user_management import register, authenticate
from crypto_utils import generate_keys
from messaging_system import send_message, receive_messages

def main():
```

```
    while True:
        print("\n=== Sistema de Mensageria Segura ===")
        print("1. Registrar")
        print("2. Login")
        print("3. Sair")
        choice = input("Escolha uma opção: ")

        if choice == '1':
            username = input("Escolha um nome de usuário: ")
            password = input("Escolha uma senha: ")
            if register(username, password):
                generate_keys(username)

        elif choice == '2':
            username = input("Nome de usuário: ")
            password = input("Senha: ")
            if authenticate(username, password):
                user_menu(username)

        elif choice == '3':
            print("Saindo...")
            break
        else:
            print("Opção inválida.")

def user_menu(username):
    while True:
        print(f"\n--- Bem-vindo, {username} ---")
        print("1. Enviar Mensagem")
        print("2. Receber Mensagens")
        print("3. Logout")
        choice = input("Escolha uma opção: ")

        if choice == '1':
            recipient = input("Nome do destinatário: ")
            message = input("Mensagem: ")
            send_message(username, recipient, message)

        elif choice == '2':
            receive_messages(username)

        elif choice == '3':
            print("Logout realizado.")
            break
        else:
            print("Opção inválida.")

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

Como Utilizar o Sistema

1. Registrar Usuários:

Execute main.py.

Escolha a opção Registrar.

Insira um nome de usuário e senha.

O sistema gerará um par de chaves (privada e pública) para o usuário.

2. Login:

Execute main.py.

Escolha a opção Login.

Insira o nome de usuário e senha.

Após o login, você poderá enviar e receber mensagens.

3. Enviar Mensagens:

No menu do usuário, escolha Enviar Mensagem.

Insira o nome do destinatário (deve estar registrado no sistema).

Digite a mensagem. A mensagem será criptografada com uma chave simétrica, a chave simétrica será criptografada com a chave pública do destinatário, e a mensagem criptografada será assinada com a chave privada do remetente.

4. Receber Mensagens:

No menu do usuário, escolha Receber Mensagens.

O sistema descriptografará a chave simétrica com a chave privada do usuário, descriptografará a mensagem e verificará a assinatura com a chave pública do remetente.

Considerações Finais

Este é um sistema básico que ilustra os conceitos de autenticação, criptografia e assinatura digital em um sistema de mensageria. No entanto, para aplicações reais, você deve considerar:

Armazenamento Seguro de Chaves: As chaves privadas devem ser protegidas adequadamente, possivelmente com senha ou em hardware seguro.

Gerenciamento de Chaves: Implementar rotinas para rotação de chaves, revogação, etc.

Transmissão Segura: Utilizar protocolos seguros (como TLS) para a transmissão de dados.

Interface de Usuário: Desenvolver uma interface mais amigável, possivelmente com uma interface gráfica ou web.

Escalabilidade e Performance: Otimizar o sistema para lidar com múltiplos usuários e mensagens de forma eficiente.

Para ilustrar o processo de autenticação com segurança em um sistema de mensageria, vamos estruturar o fluxograma com os seguintes pontos:

Processo de Autenticação: Iniciar com a entrada de credenciais do usuário e validar o token de autenticação.

Segurança e LGPD: Tratar e proteger dados pessoais e garantir a conformidade com a Lei Geral de Proteção de Dados (LGPD), armazenando apenas dados essenciais.

Vou descrever o fluxo e os principais passos para que você possa representar o fluxograma.

Fluxograma do Processo de Autenticação em um Sistema de Mensageria

1. Início:

Início do processo de autenticação.

2. Entrada de Credenciais:

Solicita ao usuário o e-mail e a senha (ou outro método seguro de identificação, como OTP ou autenticação biométrica, caso seja implementado).

3. Verificar Dados de Entrada:

Conformidade com a LGPD: Certifique-se de que o sistema apenas colete informações essenciais para autenticação, de acordo com os princípios da minimização de dados e necessidade do processamento.

Validação: Verifica se o e-mail e senha estão nos formatos corretos (ou que o OTP é válido).

4. Verificar Existência do Usuário:

O sistema consulta o banco de dados e verifica se o usuário está registrado.

Sim: Seguir para a próxima etapa.

Não: Retorna uma mensagem de "Usuário não encontrado" e encerra o processo.

5. Verificação de Senha:

Compara a senha fornecida com o hash da senha armazenada no banco de dados.

Sim: A senha está correta. Continua para a próxima etapa.

Não: Retorna uma mensagem de "Senha incorreta" e encerra o processo após várias tentativas (garantindo segurança contra ataques de força bruta).

6. Geração de Token de Autenticação:

Se as credenciais forem válidas, gera-se um token seguro (JWT ou similar) que contém informações mínimas necessárias para identificar o usuário, como ID de usuário e tempo de validade do token.

7. Armazenamento Seguro do Token no Cliente:

Recomendações LGPD: Notifica o usuário sobre o armazenamento de dados de autenticação, fornecendo

informações claras e concisas sobre como os dados de sessão serão usados e protegidos.

8. Retorno ao Usuário: O token é enviado para o cliente e armazenado com segurança (em localStorage ou cookies seguros).

Confirmação de Autenticação Concluída: Exibe uma mensagem de sucesso e permite o acesso à área segura do sistema de mensageria.

9. Fim do Processo de Autenticação.

Aspectos de Segurança e LGPD no Fluxo

Confidencialidade e Integridade: Certifique-se de que as credenciais sejam sempre criptografadas em trânsito (HTTPS) e em repouso.

Minimização de Dados: Armazene e processe apenas dados essenciais para autenticação e identifique a duração e o tipo de armazenamento de dados para o usuário.

Transparência e Consentimento: Notifique o usuário sobre o tratamento dos dados (ex.: termo de política de privacidade) conforme a LGPD, garantindo o direito de revogação do consentimento.

Representação Visual

Para o fluxograma, você pode criar cada uma dessas etapas como caixas de decisão (como "Verificar Existência do Usuário" e "Verificação de Senha") e ações (como "Geração de Token de Autenticação" e "Armazenamento Seguro do

Token"). Conecte essas caixas para guiar o usuário pelo processo até o fim da autenticação, garantindo que todos os dados sejam tratados com segurança e conformidade regulatória.