Criptografia

Cibersegurança e Defesa Cibernética

São Paulo
2025

INTEGRANTES DO PROJETO e RA'S

Daniel Baptista Acioli Vanderlei	23025608
Fábio Oliveira Spíndola	22086131
Fabrício Cândido do Nascimento	23025273
Leonardo de Souza Mouzinho	23025627

Sumário

1. Introdução

No projeto, foi adotado o algoritmo de criptografia RSA (Rivest-Shamir-Adleman) para proteger informações sensíveis dos usuários, como nome e e-mail, antes de armazená-las no banco de dados. Essa medida visa garantir maior segurança e privacidade dos dados, evitando o acesso indevido a informações pessoais como Nome e E-mail.

O RSA é um algoritmo de criptografia assimétrica, amplamente utilizado para garantir a confidencialidade e autenticidade das informações. Diferentemente da criptografia simétrica, que usa a mesma chave para cifrar e decifrar, o RSA utiliza um par de chaves:

- **Chave Pública:** usada para criptografar os dados. Pode ser compartilhada livremente.
- **Chave Privada:** usada para descriptografar os dados. Deve ser mantida em segredo.

2. Como foi utilizado o RSA no projeto

No código, a função criptografar_rsa(mensagem) é responsável por criptografar as informações pessoais do usuário antes de salvá-las no banco de dados:

```
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives import serialization
import base64

with open("public_key.pem", "rb") as f:
    public_key = serialization.load_pem_public_key(f.read())

def criptografar_rsa(mensagem):
    criptografado = public_key.encrypt(
        mensagem.encode("utf-8"),
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
    )
    return base64.b64encode(criptografado).decode("utf-8")
```

2.1. Geração das Chaves

As chaves pública e privada são geradas previamente e armazenadas em arquivos public_key.pem e private_key.pem.

A chave pública é carregada pela aplicação para criptografar os dados antes do armazenamento.

2.2. Criptografia dos Dados

Ao receber os dados do usuário (nome e e-mail) via requisição HTTP, a aplicação utiliza a chave pública para criptografar essas informações.

A criptografia é feita com o esquema OAEP (Optimal Asymmetric Encryption Padding) usando SHA-256, garantindo segurança contra-ataques de texto simples.

2.3. Armazenamento Seguro

Os dados criptografados (em formato base64 para facilitar o armazenamento) são salvos no banco de dados.

Dessa forma, mesmo que o banco seja acessado indevidamente, os dados sensíveis estarão protegidos, pois somente quem possuir a chave privada poderá descriptografá-los.

2.4. Descriptografia (Opcional)

Para acessar os dados originais, a aplicação deve usar a chave privada para descriptografar as informações.

Isso pode ser feito em um ambiente seguro e controlado, garantindo a confidencialidade.

2.5. Detalhamento do Processo

A chave pública é carregada do arquivo public_key.pem.

A mensagem (nome ou e-mail) é convertida para *bytes*.

É aplicado o *padding* OAEP (Optimal Asymmetric Encryption Padding) com SHA-256, que é uma forma segura de preenchimento.

O resultado é convertido em Base64 para facilitar o armazenamento como *string* no banco de dados.

2.6. Ponto de Uso da Criptografia na API

A criptografia acontece dentro da rota /api/prever antes do salvamento no banco:

nome = criptografar_rsa(nome)

email = criptografar_rsa(email)

3. Considerações Finais

Os benefícios da criptografia RSA incluem:

Segurança: Dados protegidos contra interceptação e acesso não autorizado.

Privacidade: Proteção da identidade e informações pessoais dos usuários.

Conformidade: Auxilia na conformidade com leis e regulamentações sobre proteção de dados.

A implementação do RSA no projeto representa uma camada importante de segurança para dados sensíveis, tornando a aplicação mais confiável e segura para os usuários.