UNIVERSIDADE DE BRASÍLIA
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

CIC0201 - Segurança Computacional - 2023.2 - Turma 01

Trabalho de Implementação 3

Gerador/Verificador de Assinaturas

Rafael Rodrigues Gama D. de Aragão - 190134780
Rodrigo Pereira Couto - 190116510

1. Introdução

Este trabalho objetiva implementar um gerador e verificador de assinaturas RSA em arquivos. São implementadas as funcionalidades de geração de chaves de 2048 bits, cifração e decifração utilizando RSA com padding OAEP, assinatura de mensagem utilizando SHA-3 e RSA OAEP e verificação de assinatura.

2. Fundamentação teórica

O RSA, nomeado em homenagem aos seus inventores Ron Rivest, Adi Shamir e Leonard Adleman, é um algoritmo criptográfico assimétrico amplamente utilizado. O algoritmo RSA envolve o uso de uma chave pública para criptografia e uma chave privada para descriptografia. A segurança do RSA é baseada na dificuldade de fatorar o produto de dois números primos grandes.

No algoritmo RSA, cada usuário tem um par de chaves: uma chave pública, que é compartilhada abertamente, e uma chave privada, que é mantida em segredo. Mensagens criptografadas com a chave pública só podem ser descriptografadas com a chave privada correspondente, e vice-versa. Isso torna o RSA uma escolha adequada para comunicação segura em uma rede insegura, pois qualquer pessoa pode criptografar uma mensagem usando a chave pública do destinatário, mas apenas o destinatário, que possui a chave privada, pode descriptografar e ler a mensagem.

O RSA com Padding Ótimo para Criptografia Assimétrica (RSA-OAEP) é uma extensão do algoritmo RSA básico projetado para aumentar a segurança ao ser usado para criptografia. O OAEP inclui uma etapa de randomização e uma função de hash para garantir que mensagens idênticas não produzam o mesmo texto cifrado, tornando-o mais resistente a certos tipos de ataques, como ataques de texto cifrado escolhido.

Uma aplicação crucial do RSA está em assinaturas digitais, onde a chave privada é usada para gerar uma assinatura digital para uma mensagem, e a chave pública correspondente é usada para verificar a autenticidade da assinatura. Para assinar uma mensagem, o remetente aplica a chave privada ao valor de hash da mensagem, criando a assinatura digital. O destinatário, usando

a chave pública do remetente, pode verificar a assinatura aplicando-a ao valor de hash da mensagem recebida. Se o valor de hash resultante coincidir com a assinatura descriptografada, a mensagem é considerada autêntica.

3. Metodologia

Para implementação do programa foi utilizada a linguagem de programação Python. O código está disponível em https://github.com/rodrigoptu12/Gerador-e-Verificador-de-assinaturas-RSA-em-Arquivos. Para executar o programa, é necessário ter uma versão do python maior ou igual à 3.8.1. O arquivo RSA.py contém a implementação das funções de geração de chaves, encriptação e desencriptação RSA. Na geração das chaves, são utilizados p e q de 1024 bits e o teste de primalidade de Miller-Rabin. O arquivo RSA_AOEP.py contém a implementação das funções de encriptação e desencriptação utilizando OAEP. No algoritmo do OAEP foi utilizado sha-1 como função de hash e mgf1 como função de geração de máscara. Além disso, para criptografia de mensagens muito grandes, foi utilizado o modo de operação em bloco ECB. O arquivo keys.py contém funções para leitura e salvamento de chaves em arquivos utilizando o padrão do openssl. Foi utilizado a biblioteca cryptography para esse fim. O arquivo sign.py contém funções para assinatura e verificação. Os arquivos são assinados usando sha3 de 256 bits como função de hash, RSA com OAEP para criptografia do hash e base64 para codificação em texto. As chaves e funções de encriptação e decriptação do nosso programa são totalmente compatíveis com o openssl, isto é, é possível encriptar uma mensagem com o nosso programa e decriptar com o openssl, o inverso também é verdadeiro. Pode-se utilizar tanto as chaves geradas pelo nosso programa quanto as do openssl. Foi adicionado ao programa uma interface de linha de comando, sendo possível executar todas as funções pelo terminal passando os comandos e arquivos. A seguir são mostrados alguns exemplos de utilização dessa interface.

Geração de Chave:

```
python main.py -genkeys
```

Criptografia:

```
python main.py -encrypt -i input.txt -o output.rsa
-privkey private_key.pem
```

Descriptografia:

```
python main.py -decrypt -i output.rsa -o output.txt
-pubkey public_key.pem
```

Assinatura:

```
python main.py -sign -i input.txt -o input.sign -privkey
private_key.pem
```

Verificação de Assinatura:

```
python main.py -verify -i input.txt -signature input.sign
-pubkey public_key.pem
```

4. Conclusão

Este trabalho propõe a implementação de um gerador e verificador de assinaturas RSA em arquivos, abrangendo diversas funcionalidades criptográficas. A fundamentação teórica fornece uma compreensão sólida do algoritmo RSA, suas propriedades e aplicações, incluindo a segurança adicional proporcionada pelo padding OAEP.

A metodologia adotada envolve o uso da linguagem de programação Python, com código disponível no GitHub para referência. As implementações abrangem a geração de chaves, cifração e decifração RSA com padding OAEP, assinatura de mensagem utilizando SHA-3 e RSA OAEP, além da verificação de assinaturas.

A compatibilidade com o OpenSSL é destacada, permitindo a interoperabilidade entre as chaves e funções do programa proposto e as do OpenSSL. A interface de linha de comando facilita a utilização do programa, proporcionando exemplos claros de como executar cada funcionalidade.

A aplicação prática do RSA em assinaturas digitais é ressaltada, demonstrando como o remetente pode assinar uma mensagem usando sua chave privada, enquanto o destinatário pode verificar a autenticidade da assinatura usando a chave pública correspondente.

Em suma, o trabalho oferece uma implementação abrangente e prática de funcionalidades criptográficas baseadas em RSA, destacando sua relevância na segurança da comunicação em redes inseguras.