Segurança Computacional - Relatório Trabalho 3

Felipe Fontenele dos Santos - 190027622

Novembro 2023

1 Introdução

Este relatório descreve a implementação de um gerador e verificador de assinaturas RSA em arquivos utilizando a linguagem de programação python. A especificação do projeto solicita a implementação de um programa com funcionalidades distintas, abrangendo a geração de chaves, cifração e decifração assimétrica RSA com a utilização do OAEP, além de um sistema de assinatura e verificação. O trabalho buscou atender a cada uma dessas áreas como geração de chaves com teste de primalidade (Miller-Rabin), cifração e decifração RSA, OAEP, formatação/parsing e a concepção de um sistema para assinatura de arquivos.

2 Desenvolvimento

2.1 Geração de chave

O arquivo 'Key.py' contém os métodos que foram utilizados para montar as chaves:

- ***two_factors(n)***: Divide um número n por 2 até que ele não seja mais divisível por 2, retornando quantas vezes foi dividido e o resultado final.
- ***miller_rabin(n, k=5)***: Implementa o teste de primalidade de Miller-Rabin para verificar se um número n é primo.
- ***generate_key(length)***: Gera uma chave aleatória com um comprimento especificado, verificando se é um número primo usando o teste de Miller-Rabin [1].
- ***generate_e(phi)***: Gera um valor e aleatório menor que phi (um parâmetro da função de Euler), sendo coprimo com phi.
- ***generate_d(e, max_value)***: Gera o inverso modular de e com relação a max_value.

- ***modular_inversion(e, max_value)***: Realiza o cálculo de inversão modular utilizando o algoritimo de Euclides Estendido, encontrando dois coeficientes que, quando multiplicados pelo número a ser invertido e o número módulo, resultam em um valor que, ao ser dividido pelo módulo, deixa um resto de 1.

- ***generate_private_and_public_key()***: Utiliza as funções anteriores para gerar chaves pública e privada, baseadas em dois números primos p e q. Calcula n = p * q, phi = (p - 1) * (q - 1), escolhe um expoente público e, e calcula o expoente privado d.

2.2 RSA

Os métodos apresentados neste conjunto desempenham os papéis necessários para o funcionamento da cifragem e decifragem RSA e OAEP do programa.

- ***integer_to_octet_string(integer, size=4)***: converte números inteiros em sequências de bytes, facilitando a representação e manipulação de dados numéricos para a codificação;

- ***rsa_cipher_decipher(message, EorD, n)***: realiza tanto a cifragem quanto a decifragem de mensagens. Ela utiliza chaves pública ou privada (EorD) e o módulo (n) para realizar operações.

- ***mask_generation_function(input_string, output_length, hash_func=hashlib.sha3_256)***: gera máscaras para proteger os dados durante o processo de codificação OAEP. Utiliza uma função de hash para criar sequências pseudoaleatórias que são cruciais na proteção dos dados.

- ***xor_bytes(a, b)***: executa a operação XOR entre dois conjuntos de bytes. É utilizada para manipular as máscaras geradas, aplicando uma operação lógica para mascarar os dados de forma determinística.

- ***oaep_cipher(public_key, message, seed=generate_key(256), label=")***: aplica o algoritmo de codificação OAEP para cifrar uma mensagem. Utiliza a geração de máscaras, operações de cifragem RSA e a manipulação de dados para aplicar a codificação OAEP.

- ***oaep_decipher(private_key, cryptogram, label=")***: decifra um criptograma utilizando o algoritmo OAEP. Realiza operações inversas à codificação do OAEP, utilizando máscaras, operações de decifragem do RSA e verificações para recuperar a mensagem original a partir do criptograma.

2.3 Implementação

Tendo os métodos anteriores em mente explicaremos o funcionamento do método que consome todos eles.

Na Main, é solicitado o nome de um arquivo o qual é feita a leitura do conteúdo do mesmo e iniciamos o fluxo de cifra, assinatura e verificação do algoritimo.

Na cifragem, é chamado um método que retorna um array contendo a chave pública e privada a serem utilizados na cifragem oaep. Nesse passo, a mensagem lida do arquivo é cifrada e decifrada e é mostrado ambas as mensagens, além da chave.

Em seguida, é realizada a assinatura da mensagem, gerando as chaves privada e pública, criando o hash da mensagem com o algoritimo SHA-3, cifrando o hash e, por fim, codificando a assinatura em base64. A assinatura é mostrada no terminal também ao final da execução.

Por fim, para realizar o terceiro passo é realizada a decodificação da assinatura (transformando a mesma em bytes) a decifragem da assinatura utilizando a chave pública gerada anteriormente no momento de realizar a assinatura e é feito uma comparação para verificar se o hash resultante da decifragem é igual ao hash original da mensagem.

3 Executando o código

Para executar o código, basta utilizar o compilador do python chamando o arquivo *main.py* na pasta raiz da seguinte maneira:

```
python3 main.py
```

Assim que o programa iniciar, ele irá solicitar o nome de algum arquivo que esteja no mesmo nível do programa para que ele realize a leitura do mesmo e faça os passos de cifrar, decifrar, gerar assinatura e validar a assinatura.

4 Conclusão

A implementação que foi feita atendeu aos requisitos principais descritos no escopo do projeto, fornecendo um conjunto completo de recursos de criptografia. A criação das chaves, que é baseada no teste de primalidade de Miller-Rabin, garante a robustez e confiabilidade do sistema, com as chaves p e q sendo primos com pelo menos 1024 bits. A utilização do algoritmo RSA, juntamente com o OAEP para criptografia assimétrica e descriptografia, garante a segurança necessária para o processo de comunicação de informações confidenciais. Além disso, a etapa de assinatura inclui o cálculo dos hashes, criptografia do hash da mensagem e formatação do resultado em BASE64 para fortalecer a integridade e autenticidade dos arquivos assinados.

Ao seguir as diretrizes estabelecidas, a implementação foi desenvolvida em conformidade com as restrições impostas. Evitou-se o uso de bibliotecas públicas específicas para primitivas criptográficas como OpenSSL, garantindo autonomia na implementação das funcionalidades essenciais.

References

[1] S. Vanstone A. Menezes P. van Oorschot. *Handbook of Applied Cryptography*. 1996.