Segurança Computacional
Implementação de um Gerador/Verificador de Assinaturas

Oseias Romeiro Magalhães / 211036123
Prof. João Gondim

04 de fevereiro de 2023

Resumo

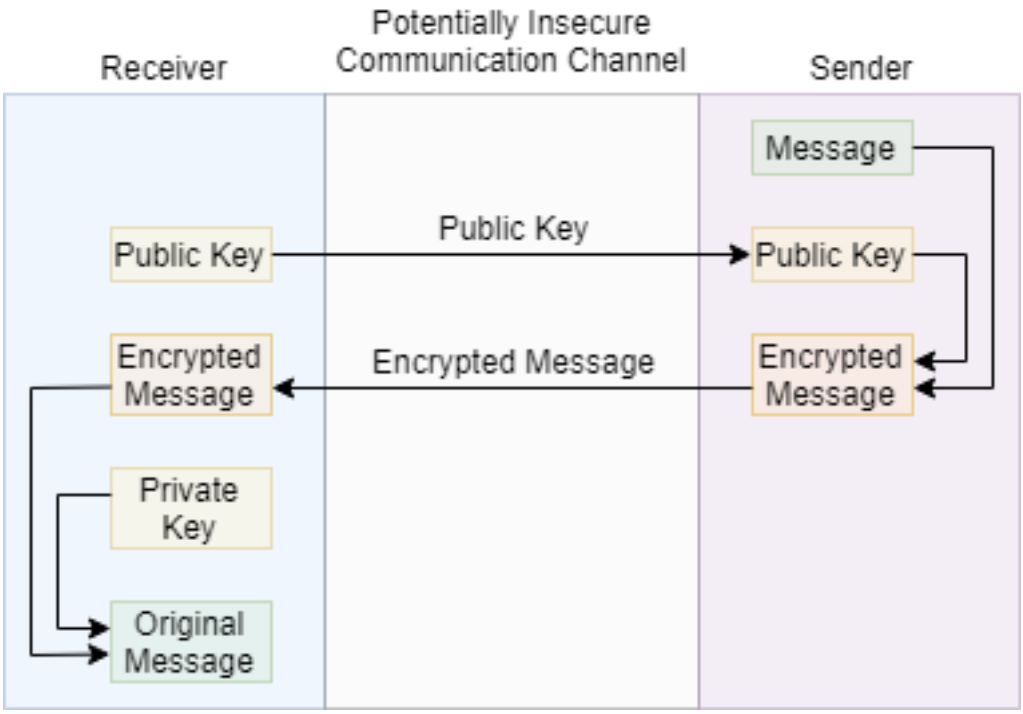
Neste trabalho de Segurança Computacional, foi implementado um Gerador/Verificador de Assinaturas em arquivos com RSA com OAEP (*Optimal asymmetric encryption padding*) com três funcionalidades: Geração de chaves, assinatura e verificação.

1 Introdução

Neste projeto, maior parte do que foi desenvolvido foi com base nos conhecimentos adquirido durando a disciplina. Tendo como base o material de Markus Kuhn [1] e complementarmente, o artigo de Yutong Zhong [7]

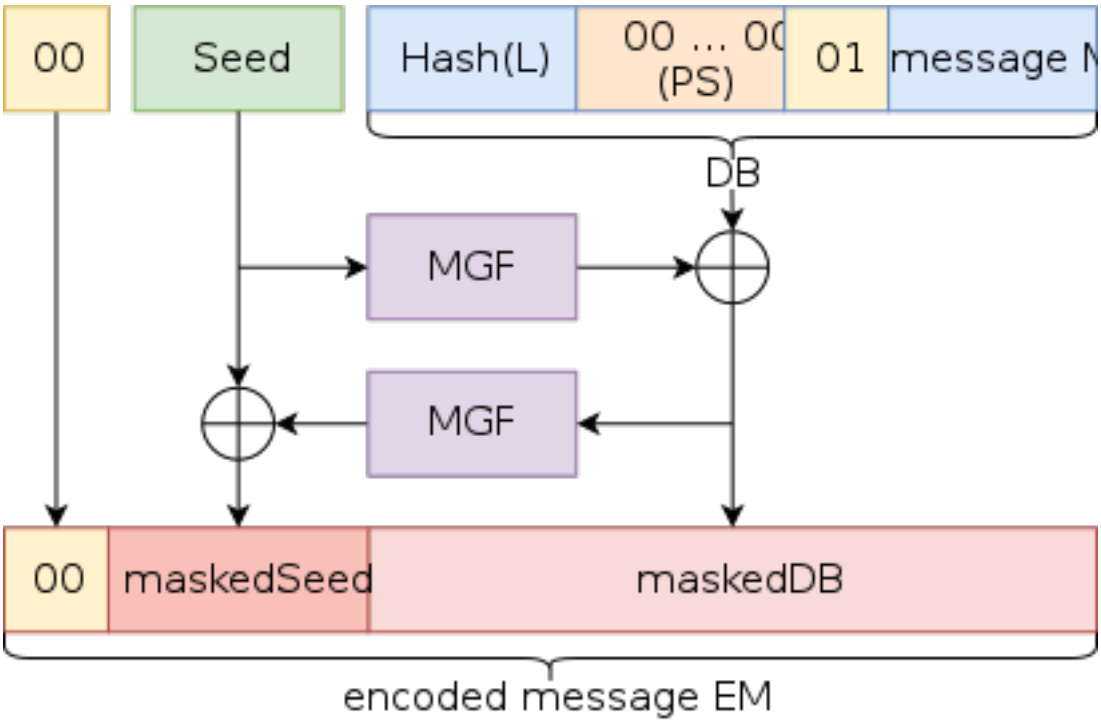
1.1 RSA

RSA é um algoritmo de criptografia assimétrica, ou seja utiliza duas chaves (uma pública e outra privada), uma que é compartilhada e outra que deve ser mantido em segredo.
Sua segurança é garantida pela dificuldade de encontrar fatores de números inteiros muito grandes, já que as chaves são compostas por dois números inteiros sendo um deles sendo um produto de dois grandes números primos (no mínimo 1024bits cada).[6]



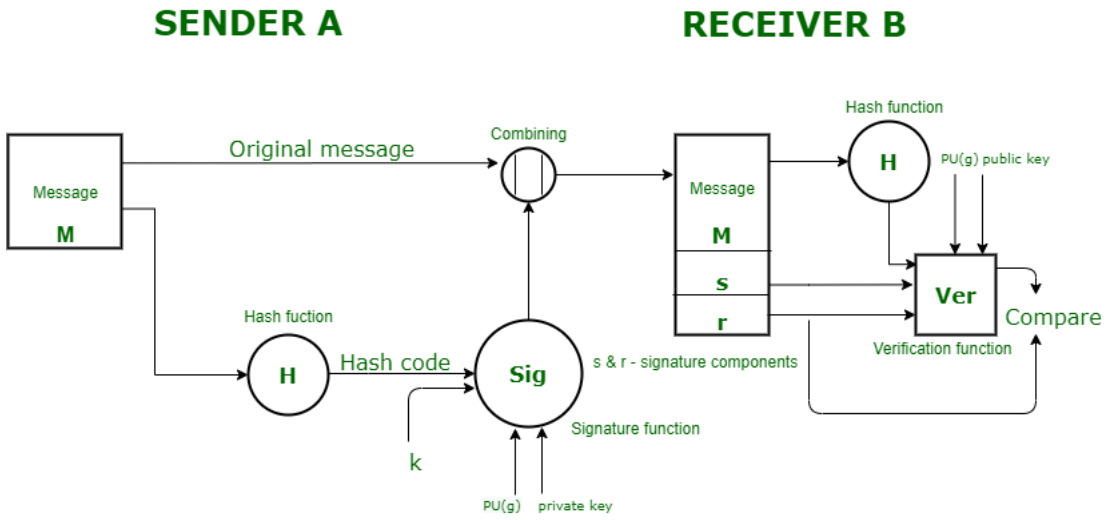
1.2 OAEP

Optimal asymmetric encryption padding é um esquema de padding utilizado juntamente com o RSA, pois sozinho não é CPA seguro. Ou seja, é vulnerável a ataques de análise em cima da mensagem criptografada e a mensagem original.[5]



1.3 DSA

Digital Signature Algorithm é um algoritmo baseado em sistema chave pública-privada, para assinatura digital de conteúdos e autenticação. Garantindo a autenticidade e integridade do conteúdo.[3]



2 Ambiente

Para o desenvolvimento desse projeto, foi utilizado a linguagem *Python* na versão 3.10.6, utilizando apenas os módulos padrão da linguagem.

3 Descrição

A estrutura adotada neste projeto foi de forma modular separar o programa nos seguintes módulos **RSA**, **OAEP** e **DSA**. Além do programa **main.py** responsável pela interação com o usuário via terminal e outros programas dentro da pasta **help** que auxiliam os módulos principais.

```
project/
├─ help/
│   ├─ CryptoMath.py
│   ├─ OAEP.py
│   ├─ primes.py
├─ resources/
│   ├─ msg.txt
├─ DSA.py
├─ main.py
├─ RSA.py
```

3.1 Run

Ao executar o programa `main.py`, é instanciado a classe `DSA` e há a interação com o usuário, dependendo do comando a ser realizado chama o método responsável.
Sendo o `DSA` responsável pelo esquema de assinatura digital, utilizando **SHA-3** para fazer hash dos documentos e com **BASE64** codificando arquivos de saída (chaves geradas e documento assinado) e decodificando arquivos de entrada (documento não-assinado e arquivo de chaves). Recebendo por herança a classe `RSA` responsável pela geração de chaves e criptografia/descriptografia com OAEP.

3.2 Geração de chaves

A geração de chaves é feita pelo método `gen()` na classe `RSA`, que utiliza do modulo `primes` para geração dos número primos mencionado na Introdução[1] e o teste Miller Rabin para verificar a primalidade dos números gerados [4]. Além do `CryptoMath` para o cálculo de modular inversa [2]. Sendo o tamanho da chave a ser gerada, definido por um atributo privado da classe (`__KEYSIZE = 1024`).

3.3 Assinatura

A assinatura do documento é feito pelo método `sign(doc_file:str, key_file:str)` da classe `DSA` que utiliza o método `encOAEP(msg:bytes, key: tuple)` da classe `RSA`. Salvando o arquivo inserido pelo usuário, assinado e com o sufixo **.signed**.

3.4 Verificação

A verificação do documento assinado é feito pelo método `vrfy(doc_signed:str, doc_original:str, key_file:str) -> bool` da classe `DSA` que utiliza o método `decOAEP(c:bytes, key: tuple)` da classe `RSA`. Após a verificação, é retornado ao usuário se o documento assinado é verdadeiro ou falso (falha de autenticação ou integridade do documento).

4 Conclusão

Assim, foi implementado um programa de geração de chaves RSA e assinatura/verificação digital de documentos. Aplicando os conhecimentos adquiridos em aula e satisfazendo os objetivos do trabalho e os requisitos de utilização de codificação em BASE64, hash com SAH-3 e teste de primalidade Miller Rabin.

Referências

[1] Markus Kuhn. *Security II: Cryptography*. https://www.cl.cam.ac.uk/teaching/1617/SecurityII/.

[2] TutorialsPoint. *Creating RSA Keys*. https://www.tutorialspoint.com/cryptography_with_python/cryptography_with_python_creating_rsa_keys.htm.

[3] wikipedia. *Digital Signature Algorithm*. https://en.wikipedia.org/wiki/Digital_Signature_Algorithm.

[4] wikipedia. *Miller–Rabin primality test*. https://en.wikipedia.org/wiki/Miller-Rabin_primality_test.

[5] wikipedia. *Optimal asymmetric encryption padding*. https://en.wikipedia.org/wiki/Optimal_asymmetric_encryption_padding.

[6] wikipedia. *RSA (cryptosystem)*. https://en.wikipedia.org/wiki/RSA_(cryptosystem).

[7] Yutong Zhong. *An Overview of RSA and OAEP Padding*. https://drpress.org/ojs/index.php/HSET/article/view/431.