Universidade de Brasília - Instituto de Ciências Exatas
Departamento de Ciência da Computação

Segurança Computacional – Trabalho 3

Cifra RSA-OAEP

Prof. João Gondim
outubro de 2021

Alunos:
Vitor Vasconcelos de Oliveira
Álvaro Veloso Cavalcanti Luz

Matrículas:
180114778
180115391

1 - Informações relevantes sobre o Programa:

- Sistema Operacional usado: Windows 10
- Compilador/linguagem: Python 3.9.0
- IDE utilizada: Visual Studio Code (funcional sem a IDE também)

2 - Execução do Programa:

- Para executar o programa basta abrir o prompt de comando no diretório onde o arquivo do programa está armazenado e executa-lo com “py trabSC3.py” ou “python3 trabSC3.py”
- Após a execução do programa, surgirá um menu de interação com o programa, basta seguir com as informações pedidas de acordo com as opções selecionadas.

- OBS 1: serão criados arquivos .txt no diretório do programa, por isso é recomendado uma pasta específica para roda-lo
- OBS 2: Devido ao fato de que a geração de novos números primos aleatórios pode ser um processo demorado, foi incluído, no processo de encriptação da mensagem, a opção de reutilizar chaves geradas em sessões anteriores. As chaves geradas são salvas em um arquivo ".txt" cujo o nome deve ser escolhido pelo usuário toda vez que novas chaves forem geradas.
- OBS 3: Toda vez que se seleciona uma chave para a encriptação de uma mensagem, é gerado um arquivo "mensagem<nome da chave>.txt" que contém a assinatura da mensagem e a mensagem encriptada. Sempre que uma chave for reutilizada, o conteúdo do arquivo de mensagem correspondente a essa chave é sobrescrito.

3 - Problema Abordado:

Este projeto busca abordar uma implementação em python 3.9 de um gerador e verificador de assinaturas RSA em arquivos. A criptografia RSA é amplamente utilizada nos sistemas de encriptação de mensagens. O algoritmo em questão utiliza-se de um par de chaves, uma pública (utilizada para cifrar a mensagem) e uma privada (que é a única capaz de decrepitar a mensagem cifrada).

A ampla utilização do RSA em aplicações se dá devido ao fato de que este é um algoritmo de encriptação facilmente computável, cujo a decriptação é extremamente custosa computacionalmente, exceto que se tenha informação especial. Para que isto seja possível, o método se baseia na multiplicação e fatoração. A fatoração é o processo de separar um número "n" nos primos que multiplicados entre si resultam em "n". Apesar de ser conceitualmente simples, à medida que "n" fica maior, a quantidade de tempo necessária para a realização da fatoração aumenta exponencialmente. Tendo isso em mente, o método se baseia em definir dois números primos aleatórios que serão multiplicados entre si, e o resultado dessa multiplicação será utilizado para então codificar uma mensagem com exponenciação modular. Dessa forma, pode-se definir o produto destes números como uma chave de acesso público e armazenar os fatores primos para fazer a decodificação.

Esclarecida a fundamentação teórica que justifica a existência do algoritmo, tem-se a seguinte divisão de passos para seu funcionamento interno:

- Geração de dois números primos grandes "p1" e "p2"
- Cálculo de um valor N tal que N = p1 * p2, ou seja, de modo que "N" seja o produto de p1 e p2;
- Definição da função ϕ ("phi") de N. A função ϕ(x) é uma função que retorna a quantidade de números que não são fatores de x. É importante ressaltar que ϕ(x) = x-1 para todo valor x primo, e que é válida a propriedade ϕ(a*b) = ϕ(a) * ϕ(b) . Logo, aplicando em N, ϕ(N) = ϕ(p1) * ϕ(p2) = (p1-1) * (p2-1);
- Seleciona-se um inteiro "e" um número ímpar que seja co-primo de ϕ(N), ou seja, que ambos sejam divisíveis apenas por 1;
- Por fim calcula-se um valor d para o qual d = (1 mod(ϕ(N)))/e.

Feito isso, tem-se então todos os dados necessários para se encriptar e decriptar mensagens, sendo o par (N,e) a chave pública e o par (N,d) a chave privada.

A encriptação de uma mensagem M baseia-se em usar a chave pública para executar o seguinte cálculo de exponenciação modular: C = (M^e) mod(N), sendo o valor C resultante a mensagem encriptada. Para decriptá-la usa-se a chave privada, no seguinte cálculo: M = (C^d) mod(N) onde C é a mensagem encriptada, e M é a mensagem original.

Em complemento ao algoritmo RSA, utilizamos o modelo OAEP(Optimal Asymmetric Encryption Padding) para trazer um fator de aleatoriedade ao algoritmo e evitar a decifração parcial da mensagem encriptada e outras formas de vazamento de informação. Aplicamos ele antes da encriptação da mensagem e depois da decriptação com RSA. Esse modelo adiciona ao RSA tradicional o pré-processamento da mensagem usando duas funções de hash: **G**, com output de "g" bits e **H**, com output de "h" bits, além de uma seed randômica "**s**". Concatena-se uma mensagem **M** com caracteres "0" até que tenha tamanho igual a "g" gerando **M'**. Então, em seguida, aplica-se um **XOR** bitwise entre **M'** e **G**(**s**) e concatenamos isso com outro **XOR** bitwise entre **s** e **H**(**M'**$\oplus$ **G**(**s**)) gerando um uma mensagem mascarada **X**. Por fim, temos que **X = M'**$\oplus$**G**(**s**) || **H**(**M'**$\oplus$**G**(**s**)).

4 - Elaboração do código:

Para a elaboração do código foi utilizada a metodologia de pair programming para a divisão de tarefas no desenvolvimento.

Foi escolhido um modelo com três entidades principais para a aplicação do algoritmo RSA **Server**, **Sender** e **Receiver.** O **Receiver** representa a entidade que recebe a mensagem, é responsável por calcular várias chaves privadas e públicas, uma de cada para toda entidade que deseja se relacionar, mantendo a privada para si e disponibilizando a pública no **Server**. Além disso, ele recebe a mensagem encriptada e realiza sua decriptação, posteriormente testando se a *Assinatura* recebida ao ser decriptada com a chave pública do **Sender** é válida, garantindo que a mensagem foi recebida sem alterações.

Sender representa a entidade que envia a mensagem, possuindo por questão de simplicidade, chaves pública e privada constantes. É o responsável por criar, encriptar e enviar a mensagem após receber a chave pública gerada pelo **Receiver**, também é onde gera-se a *Assinatura* através de um processo de cifragem e *hashing* da mensagem com sua própria chave privada.

Por último temos o **Server** responsável por armazenar a mensagem cifrada e a *Assinatura* em arquivos ".txt" e por mediar o envio de dados entre as outras duas entidades, seja da chave pública enviada pelo **Receiver** ao **Sender** ou da mensagem encriptada, da *Assinatura* e da chave pública enviada do **Sender** ao **Receiver**. Essas entidades foram todas representadas por classes no código-fonte.

Além dessas três classes, outras quatro foram criadas com o intuito de auxiliar as demais já relatadas em seus processos. As duas primeiras são as classes de **Encrypt**, que realizam apenas o processo de encriptação da mensagem usando o algoritmo RSA-OAEP, e **Decrypt**, que realiza o processo de decriptação da mensagem também com o RSA-OAEP. Vale ressaltar que ambas classes apenas realizam o cálculo de encriptação recebendo todos os parâmetros como a mensagem e as chaves já prontos.

As duas últimas classes são **KeyGenTools** e **CryptoTools** apresentam diversas funções desenvolvidas para auxiliar no processo de geração das chaves e

criptografia respectivamente. Em **KeyGenTools** observa-se métodos que auxiliam na criação dos números primos com 1024 bits, o teste de primalidade de Miller e o método Euclidiano para MDC. Já em **CryptoTools**, seus métodos auxiliam em ambos os processos de encriptação e decriptação. Esta classe possui funções de : conversão de bytearray para inteiro, criação de hash da mensagem usando o sha3, geração de máscara para a encriptação da mensagem e a operação de XOR bitwise para bytearrays.

5 - Resultado e Considerações finais:

Finalizando, pode-se considerar que o objetivo central do trabalho foi cumprido com êxito. O RSA foi implementado e simulado adequadamente, seguindo todas as diretrizes delimitadas na especificação, tais como a: geração de chaves com teste de primalidade (Miller-Rabin), a cifração e decifração RSA, uso do OAEP no processo de criptografia e formatação ou parsing da mensagem.

As maiores dificuldades encontradas no desenvolvimento do trabalho se resumem, em sua maioria, à complexidade de montar uma estrutura que pudesse conciliar a funcionalidade do código como uma ferramenta de criptografia e ao mesmo tempo replicar como seria um cenário de aplicação dessa ferramenta na realidade.

Possíveis futuras melhorias no projeto poderiam ser feitas em diversas frentes, como: melhorias na interface (fazendo uso de uma biblioteca de interface gráfica para tornar a ferramenta mais atrativa), aplicação do código desenvolvido no processo de comunicação de um servidor, além de alterações ao código que possibilitem que haja uma comunicação do **Receiver** com vários **Senders** e melhorias no sistema de gerenciamento de mensagens e chaves.

6 - Referências:

- https://www.youtube.com/watch?v=wXB-V_Keiu8
- https://en.wikipedia.org/wiki/RSA_(cryptosystem)
- https://www.youtube.com/watch?v=ZwPGE5GgG_E&ab_channel=ArtoftheProblem

Obrigado!!!