Universidade de Brasília

Departamento de Engenharia Elétrica

Segurança Computacional
Trabalho 2 - RSA-OAEP

Autor:

Hugo Silva de Vasconcelos — 180102028

Brasília
11 de fevereiro de 2023

Conteúdo

1 Definições

1.1 RSA

É um algoritmo de criptografia assimétrica que permite que mensagens sejam codificadas com uma chave e decodificadas com uma outra chave distinta. O algoritmo utiliza duas chaves: uma chave pública, conhecida por todos, e uma chave privada, conhecida apenas pelo destinatário. O remetente codifica a mensagem usando a chave pública, e o destinatário a decodifica usando sua chave privada, ou vice-versa. As chaves são relacionadas entre si por um método matemático que, tendo conhecimento de apenas uma chave, a probabilidade de se descobrir a outra é próxima de 0. A segurança do algoritmo é baseada na dificuldade na geração e fatoração de números grandes originados a partir de números primos.

1.2 RSA-OAEP

O RSA-OAEP (Optimal Asymmetric Encryption Padding) é uma versão do algoritmo RSA de criptografia que inclui um processo de preenchimento (padding) ótimo para aumentar a segurança da criptografia. Em vez de simplesmente codificar a mensagem, o RSA-OAEP adiciona informações extras (preenchimento) antes da codificação com o objetivo de tornar a mensagem mais difícil de ser decifrada por atacantes. Além disso, o RSA-OAEP inclui uma função hash para garantir a integridade da mensagem e uma função de preenchimento (padrão) para ajudar a impedir ataques. Em resumo, o RSA-OAEP é uma versão mais segura do algoritmo RSA para criptografia de chave pública.

2 Funções

2.1 isPrime

Verifica se o número inserido é primo realizando os seguintes testes:

1. (Caso o número esteja, este é primo)
2. Verifica se o número inserido é divisível por algum dos primos na lista, caso seja, este número não é primo
3. Verifica pelo teste de Miller Rabin

2.2 miler_rabin

Implementa o teste probabilístico de miller rabin de forma fatorada para aumentar o desempenho do algoritmo e permitir a geração de chaves de até 2048 bits.

Fontes: (`https://youtu.be/zmhUlVck3J0`; `https://youtu.be/-BWTS_1Nxao`)

2.3 generateLargePrime

gera números aleatórios de tamanho em bits igual ao da chave e verifica se o mesmo é primo. Em caso positivo retorna o mesmo

2.4 generateKeys

Gera p e q primos de tamanho igual ao tamanho da chave, calcula $N = p \times q$, calcula o totiente de Euler $(phiN) = (p-1) \times (q-1)$. Gera um número e de tamanho igual a keysize e coprimo de phiN, esta será a chave pública. Depois, gera um número d sendo o modulo inverso de phiN, esta sera a chave privada.

2.5 isCoprime

Utiliza a função gcd(Greatest Common Divisor) para calcular o maior divisor comum e verificar se ambos os números inseridos são coprimos

2.6 modularInverse

Utiliza o algoritmo de Euclides para determinar o módulo inverso entre dois números

2.7 rsa_encrypt

Cifra a mensagem inserida caractere por caractere utlizando a função matemática $c = m^e mod(N)$ onde c é a cifra do caractere m e N é $p \times q$ calculado na função generateKeys e e é a chave pública da cifração

2.8 rsa_decrypt

Decifra a mensagem cifrada inserida caractere por caractere utlizando a função matemática $m = c^d mod(N)$ onde c é a cifra do caractere m e N é $p \times q$ calculado na função generateKeys e d é a chave privada da cifração.

2.9 oaep_mask

Implementa a função Optimal Asymmetric Encryption Padding de acordo com o bloco mais a esquerda da Figura 1. Onde a mensagem original é incrementada por uma sequencia de m bits onde $m = keysize - k0$, keysize, é o tamanho da chave em bits, e k0 é o tamanho da hash gerada na função H.

Após a mensagem sofrer esse padding, ela passa por um xor com o resultado de G, sendo G a função hash do número aleatório r, gerando então a primeira parte P_1 do resultado da função mask. Esse P_1 também entra na função hash H e é feito um xor com o número r, gerando então a segunda parte da mensagem P_2.

O resultado da função oeap_mask é $P_1 + P_2$

2.10 oaep_unmask

Já na função unmask o processo é similar ao da função mask descrito acima, porém inverso. A mensagem cifrada é a entrada da função, e essa é dividida em duas partes, P_1 e P_2.

A parte P_2 passa inicialmente por um xor com a função hash H de P1, gerando o número r. Que por sua vez entra na função hash G e o resultado passa por um xor com P1, descobrindo então a mensagem com padding.

Daí, se remove o padding da mensagem e obtêm-se a mensagem original.

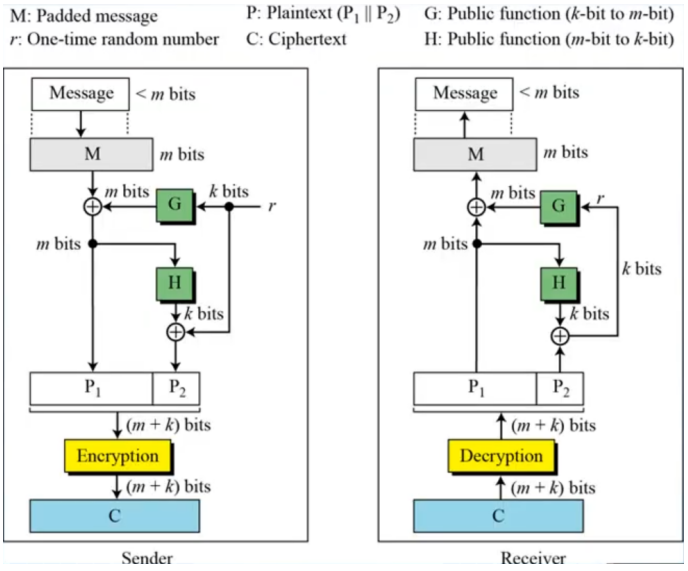

Figura 1: Diagrama de Blocos das funções oaep_mask e oaep_unmask

2.11 main

Por fim, na função main, utiliza-se um menu para organizar as opções do algoritmo. Na opção 1, são geradas as chaves e é dada a opção de salva-las em arquivos para garantir sua persistência.

A opção 2 por sua vez lê essas chaves e o módulo desses arquivos gerados em 1 e os carrega para o programa.

Já na opção 3, é gerada a assinatura de um arquivo. Nessa opção, é feito um hash dos bytes lidos do arquivo e esse hash é cifrado com o rsa-oaep, codificado em base64 e gravado em um arquivo, de mesmo nome do original acrescido de _sign no fim.

Por fim, na opção 4 é realizada a validação da assinatura de um arquivo. Essa validação é realizada da seguinte maneira. O arquivo de assinatura é lido, carregado para o programa e decifrado. Então é realizado um novo hash do arquivo a ser validado. Por fim, a assinatura decifrada é comparada ao hash obtido e em caso de igualdade uma mensagem informa que a assinatura foi validada. Em caso de diferença entre os dois, uma mensagem de arquivo corrompido aparece.

3 Limitações

Até onde foi observado nos testes do algoritmo, a única limitação apresentada foi no momento da verificação da assinatura, onde o parâmetro P1_Size da função oaep_unmask é fornecido pela função oaep_mask executada na opção 3 dessa forma, a validação da assinatura só pode ocorrer na mesma execução do código em que a assinatura foi feita. Caso contrário, o valor se perde, e um erro é exibido.

4 Referência

Y. Feng, William. How To Tell If A Number Is Prime: The Miller-Rabin Primality Test: `https://youtu.be/zmhUlVck3J0`

Y. Feng, William. How to Implement the Miller-Rabin Primality Test Test: `https://youtu.be/-BWTS_1Nxao`

Sekhar Sanaboina, Chandra. OAEP in RSA: `https://youtu.be/zi7gfind7mE.`