# Segurança Computacional

## CIC0221 – 2022/1

## Trabalho 2

## Universidade de Brasília - 19/09/2022

**Marcus Vinicius Oliveira de Abrantes 190034084**

**João Antonio Desiderio de Moraes 160126975**

## Introdução

Neste trabalho estará descrito a implementação conjunta dos algoritmos de criptografia RSA-OAEP e AES. Os algoritmos serão usados para simular um gerador e verificador de assinaturas. As seções estão divididas de forma a especificar o funcionamento de cada algoritmo e em que arquivo se encontram no código.

## Implementação

O trabalho foi desenvolvido na linguagem Java. Essa opção foi feita devido a maior facilidade de manutenção e debugging do projeto. Ocorreu versionamento por Git utilizando a plataforma GitHub.

É possível acompanhar os principais processos executando a classe src/main/java/br/com/segcomp/App, onde o método Main demonstra os processos de cifração e decifração por AES, obtenção do hash da mensagem e da chave, cifração e decifração por RSA e compartilhamento por OAEP.

## RSA

RSA é um algoritmo de chave pública amplamente utilizado para transmissão de informações de forma segura. Se baseia em conceitos matemáticos de modularização. O Funcionamento é descrito da seguinte forma:

Uma entidade pública precisa de dois números primos grandes, de forma que a fatoração de um número que possua esses dois números como dividendos seja extremamente custosa computacionalmente.

A partir desses dois números p e q são efetuadas tais operações matemáticas:

É calculado n = p * q.

É calculado a Função Totiente de Euler dada por phi(n) = (p-1)*(q-1)

Um inteiro e é escolhido tal que 1<e<phi(n) e e e phi(n) sejam coprimos, isto é, não possuam divisores em comum.

Calcule um d tal que (d*e) mod phi(n) = 1;

Após estabelecidos todos os valores, serão determinados como chave pública da entidade os valores e e n.

Já a chave privada é definida pelos valores d e n.

A criptografia pelo algoritmo RSA se dá pelas seguintes etapas

O valor a ser criptografado m é elevado à potência de d módulo n = C.

C então pode ser descriptografado ao fazer-se c elevado à potência de e modulo n.

Se todos os passos foram executados corretamente, pode-se garantir que ((m^d) mod n)^e mod n = m. Assim é possível obter a mensagem inicial.

No código enviado, este algoritmo estará descrito em dois arquivos: PrimeGenerator.java, responsável por gerar 2 números primos de 1024 bits; RSA.java que faz subsequentemente as operações de criptografia e descriptografia, segundo os passos descritos em 2.*.

Para a geração de número primos foi utilizado o algoritmo Millher-Rabin na qual um número de 1024 bits aleatório é verificado como primo a partir de fatores probabilísticos. De forma simplificada, para um número ímpar N é subtraído 1, e então verificado quantas k vezes o número precisa ser dividido por 2 até que chegue a um inteiro ímpar, este inteiro d também será usado nos cálculos. Após isso, um inteiro a < n é escolhido, em seguida é feito b = a ^d módulo n. Caso b seja igual a 1 mod n ou –1 mod n, n é um possível primo. Após isso durante k etapas, será feito b^2 mod n, desta vez em diante, caso o resultado seja 1 mod n então n com certeza não é primo. Caso seja igual a –1 mod n, n é um provável primo. Os passos descritos foram implementados na função millerRabin em PrimeGenerator.java.

## OAEP

Uma extensão ao algoritmo RSA é feita pelo algoritmo OAEP, responsável por fazer padding da mensagem. O Seu funcionamento foi inserido no arquivo OAEP.java, que também necessita do arquivo MGF1.java, que faz a mascaragem dos bits, parte do processo do OAEP.

O OAEP utiliza de uma função de HASH e uma função geradora de máscara, que no caso deste projeto é a MGF1. No algoritmo é possível criptografar um mensagem junto a um label. O hash do label é concatenado a um sequência de zeros que preenche espaços vazios junto ao byte 1 e à mensagem formando DB. Também é definido uma Seed, que passa pela máscara MGF1 para então ser transformada em uma operação XOR com DB resultando em maskedDB, que também passar pela máscara, também sendo usado em um XOR com a Seed, formando uma maskedSeed. Em uma sequência de bytes final será salvo o byte 0, seguido de maskedSeed, por fim maskedDB.

A partir das operações inversas, é possível obter os valores iniciais usados no padding. Em OAEP.java estão implementadas as etapas descritas.

## AES

Conforme especificado, buscamos a implementação do padrão de criptografia AES - Advanced Encryption Standard, modo Counter. Optamos pela implementação da versão de 128 bits para a chave e os blocos cifrados.

O processo segue todos os passos de cifração e decifração definidos no padrão. Para a versão de 128 bits, são organizados dez rodadas de transformação. Antes da rodada inicial, a chave é adicionada ao bloco por meio de uma operação XOR bit-a-bit. Nas próximas nove rodadas, o bloco passa pelos métodos: - SubBytes: substituição dos bytes usando uma tabela SBox, que mapeia cada valor ao resultado de uma série de operações sobre ele considerado o campo de Galois 2^8;
- ShiftRows: deslocamentos de bytes no bloco; - MixColumns: combinação das colunas por meio de uma multiplicação matricial;
- AddRoundKey: adição da chave da rodada por meio de XOR bit-a-bit. A última rodada é similar as anteriores, porém sem executar o procedimento MixColumns.

Para a decifração, todos as etapas são aplicadas em sua forma inversa.

O protocolo determina que a chave deve passar por um processo de expansão para ser aplicada a todas as rodadas. O processo consistem em tomar a chave como words de 32 bits, e gerar uma nova word de chave a partir das words já disponíveis até completar as 11 chaves necessárias. A geração envolve substituições, rotações e operações XOR entre as words. A implementação dessa etapa está contida na classe AES128KeyScheduler.

Da forma como foi descrito até aqui, o padrão é capaz de cifrar e decifrar um bloco de 128 bits. Poderíamos utilizar o mesmo processo seguidas vezes para cifrar cadeias de 128 bits. O resultado apresentaria uma vulnerabilidade relevante: como a mesma chave foi utilizada para todos os blocos da cadeia, blocos iguais gerariam cifrações iguais. Essa característica abre portas para ataques por análise.

O modo CTR, ou Counter, visa aumentar a segurança do protocolo. É utilizado um contador como vetor de inicialização das chaves, ou seja, a cada bloco é cifrado pela chave operada com XOR ao valor do contador, permitindo que a chave varie ao longo da cifração da cadeia.

A seguir, fazemos menção a peculiaridades interessantes da implementação.

Os métodos operam sobre representações de blocos de bytes encapsulado na classe Block. A classe possui métodos para gerar blocos de n bits a partir de cadeias maiores. Também implementamos uma sobrecarga do método toString para facilitar a impressão dos blocos na tela.

A chave inicial é gerada através da biblioteca SecureRandom, nativa do Java. A chave expandida, por sua vez, resulta de um processo totalmente implementado pelos alunos.

Os métodos MixColumns e InvMixColumns implementam as transformações necessárias corretas, porém não são uma tradução literal do processo teórico que as explica. Isso ocorreu pois as operações no campo de Galois envolvem transformações polinomiais complexas de ser implementadas. Para fins didáticos, buscamos alternativas mais simples de implementação. No caso do InvMixColumns, usamos tabelas de equivalência para obter os resultados corretos.