**Trabalho 3 de Segurança Computacional RSA-HASH-OAEP**

**Douglas Vieira Alcantara – 18/0031333**

# OAEP

O Optimal Asymmetric Encryption Padding (OAEP) é um esquema de preenchimento utilizado em criptografia assimétrica, especialmente em algoritmos como o RSA. Seu objetivo principal é melhorar a segurança e a eficácia do processo de codificação, tornando-o mais resistente a ataques.

## Como Funciona o OAEP?

1. **Codificação da Mensagem:**
   * A mensagem original é expandida e combinada com um rótulo (label) usando uma função hash.
   * Um valor aleatório (chamado de "seed") é gerado.
2. **Máscara Geradora de Máscara (MGF):**
   * Uma função chamada MGF é aplicada à semente para criar uma máscara que será usada para obscurecer parte da mensagem.
3. **Aplicação das Máscaras:**
   * A máscara gerada pela MGF é aplicada à parte da mensagem.
   * A semente também é obscurecida usando a parte da mensagem restante.
4. **Combinação Final:**
   * As partes obscurecidas da mensagem e da semente são combinadas para formar a mensagem codificada final.
5. **Codificação RSA:**
   * A mensagem codificada é então passada pelo processo de codificação RSA usando a chave pública.

## Objetivos do OAEP:

* **Confidencialidade:** Evita padrões previsíveis na mensagem codificada.
* **Integridade:** Introduz uma verificação de integridade através do uso de funções hash.
* **Segurança contra Ataques:** Minimiza riscos de ataques.

## Vantagens do OAEP:

* **Segurança Aprimorada:** OAEP oferece um nível mais alto de segurança contra ataques como o "ataque de texto cifrado escolhido" (chosen-ciphertext attack).
* **Proteção contra Ataques Adicionais:** Ajuda a mitigar ataques de injeção de dados e ataques à integridade da mensagem.

Em resumo, OAEP é uma técnica de preenchimento que visa fortalecer a segurança da criptografia assimétrica, com confidencialidade, integridade e resistência contra diversos tipos de ataques. Ele é amplamente utilizado em algoritmos como o RSA para garantir uma proteção mais robusta dos dados transmitidos.

# Assinatura RSA

A assinatura RSA é uma técnica criptográfica que utiliza algoritmo de criptografia assimétrica RSA para prover autenticidade, integridade e não alteração de dados digitais. Ela permite que o remetente de uma mensagem prove a autoria da mensagem e que o conteúdo não foi alterado durante a transmissão.

## Como Funciona a Assinatura RSA:

1. **Geração de Chaves:**
   * O remetente gera um par de chaves RSA: uma chave privada para assinar e uma chave pública para verificar a assinatura.
2. **Hash da Mensagem:**
   * Uma função hash é aplicada à mensagem original para produzir um valor resumido (hash) fixo.
3. **Assinatura:**
   * O remetente criptografa o hash gerado com sua chave privada, gerando a assinatura digital. Isso garante a autenticidade, pois apenas o possuidor da chave privada poderia ter produzido essa assinatura.
4. **Verificação da Assinatura:**
   * O destinatário recebe a mensagem, calcula o hash da mensagem recebida e descriptografa a assinatura usando a chave pública do remetente.
   * Se o hash calculado da mensagem original coincidir com o hash descriptografado da assinatura, a verificação é bem-sucedida.

## Objetivos da Assinatura RSA:

* **Autenticidade:** A assinatura digital garante a autoria da mensagem, indicando que ela foi realmente enviada pela entidade que possui a chave privada dela.
* **Integridade:** Qualquer alteração na mensagem resultará em uma assinatura inválida.

## Vantagens da Assinatura RSA:

* **Segurança Assimétrica:** Utiliza a robustez matemática do algoritmo RSA para garantir segurança.
* **Padrão Amplamente Adotado:** É uma técnica de assinatura digital amplamente aceita e utilizada em diversos protocolos de segurança.

# Documentação do Código

## 1. main()

Objetivo: - A função principal que orquestra a execução do programa.

Funcionalidades: 1. Leitura de um arquivo original para assinar. 2. Geração de pares de chaves pública e privada. 3. Codificação e decodificação de uma mensagem usando o esquema de padding OAEP. 4. Geração e verificação de uma assinatura digital. 5. Checagem de integridade de um arquivo original e de um arquivo modificado.

## 2. Funções do Programa

### 2.1 checar\_assinatura(hash\_arquivo\_assinado, hash\_arquivo\_decifrado\_com\_chave)

Objetivo: - Verificar se a assinatura digital é válida comparando dois hashes.

Parâmetros: - hash\_arquivo\_assinado: Hash do arquivo original. - hash\_arquivo\_decifrado\_com\_chave: Hash decifrado usando a chave privada.

Retorno: - Retorna 1 se a verificação for bem-sucedida e 0 em caso de erro.

### 2.2 ler\_arquivo\_modo\_binario(nome\_arquivo)

Objetivo: - Ler um arquivo binário e retornar seus dados.

Parâmetros: - nome\_arquivo: Nome do arquivo a ser lido.

Retorno: - Retorna os dados do arquivo lido.

### 2.3 gerar\_chaves()

Objetivo: - Gerar pares de chaves pública e privada para o algoritmo RSA.

Retorno: - Retorna as duas chaves

### 2.4 RSA\_codificar(mensagem\_codificada, chave\_publica)

Objetivo: - Codificar uma mensagem usando o algoritmo RSA.

Parâmetros: - mensagem\_codificada: Mensagem a ser codificada. - chave\_publica: Chave pública para codificação.

Retorno: - Retorna a mensagem codificada.

### 2.5 RSA\_decodificar(mensagem\_codificada, chave\_privada)

Objetivo: - Decodificar uma mensagem codificada usando o algoritmo RSA.

Parâmetros: - mensagem\_codificada: Mensagem codificada. - chave\_privada: Chave privada para decodificação.

Retorno: - Retorna a mensagem decodificada.

### 2.6 formar\_bloco\_dados(l\_hash, mensagem, k, tam\_hash)

Objetivo: - Formar um bloco de dados para o esquema de padding OAEP.

Parâmetros: - l\_hash: Hash do rótulo. - mensagem: Mensagem a ser inserida no bloco de dados. - k: Tamanho do bloco de dados. -tam\_hash: Tamanho do hash.

Retorno: - Retorna o bloco de dados.

### 2.7 OAEP\_codificar(mensagem, chave\_publica, tam\_hash, rotulo="", k=128)

Objetivo: - Codificar uma mensagem usando o esquema de padding OAEP.

Parâmetros: - mensagem: Mensagem a ser codificada. - chave\_publica: Chave pública para codificação. - tam\_hash: Tamanho do hash. -rotulo: Rótulo para o hash. - k: Tamanho do bloco de dados.

Retorno: - Retorna a mensagem codificada.

### 2.8 OAEP\_decodificar(mensagem\_codificada, chave\_privada, tam\_hash, rotulo="", k=128)

Objetivo: - Decodificar uma mensagem codificada usando o esquema de padding OAEP.

Parâmetros: - mensagem\_codificada: Mensagem codificada. - chave\_privada: Chave privada para decodificação. - tam\_hash: Tamanho do hash. - rotulo: Rótulo para o hash. - k: Tamanho do bloco de dados.

Retorno: - Retorna a mensagem decodificada.

### 2.9 mgf(seed, tam\_mascara, hash\_func)

Objetivo: - Máscara de geração de função usando hash.

Parâmetros: - seed: Semente para a máscara. - tam\_mascara: Tamanho da máscara desejada. - hash\_func: Função de hash a ser utilizada.

Retorno: - Retorna a máscara.

### 2.10 xor(a, b)

Objetivo: - Operação XOR entre duas sequências de bytes.

Parâmetros: - a: Sequência de bytes. - b: Sequência de bytes.

Retorno: - Retorna o resultado da operação XOR.

### 2.11 mdc\_estendido(a, b)

Objetivo: - Algoritmo estendido de Euclides para encontrar o máximo divisor comum.

Parâmetros: - a: Primeiro número. - b: Segundo número.

Retorno: - Retorna uma tupla (mdc, x, y) onde mdc é o máximo divisor comum e x, y são coeficientes inteiros tais que ax + by = mdc.