UnB Instituto de Ciências Exatas Departamento de Ciência da Computação

Segurança Computacional - 2025/1

Professora Priscila Solis

João Victor Prata (202028857), Nikolas Negrao (202024722)

Relatório Trabalho de Implementação 2 - Gerador / Verificador de Assinaturas

Este relatório tem como objetivo descrever sobre a aplicação e funcionamento do sistema de geração e verificação de assinaturas digitais baseado no algoritmo RSA com preenchimento OAEP.

Descrição dos Arquivos:

- **rsa_signature_tool.py**: arquivo Python contendo o código fonte completo da implementação das primitivas de geração de chave, assinatura, verificação e codificação/decodificação OAEP.
- **mensagem.txt**: arquivo de texto contendo uma mensagem de exemplo que será assinada e em seguida verificada. Serve como entrada para os testes de assinatura digital.
- **mensagem_assinada.txt**: arquivo gerado pelo sistema após a assinatura da mensagem. Contém tanto a mensagem original (codificada em Base64) quanto a assinatura digital, estruturadas com marcadores (*BEGIN/END MESSAGE* e *BEGIN/END SIGNATURE*), possibilitando a verificação de autenticidade.

Funcionamento da OAEP:

OAEP (Optimal Asymmetric Encryption Padding) é um esquema de preenchimento utilizado em criptografia assimétrica, especialmente com RSA, com o objetivo de aumentar a segurança da cifragem ao evitar ataques baseados em estruturas determinísticas.

Na prática, o RSA puro possui uma limitação: a mesma mensagem sempre gera o mesmo texto cifrado, o que facilita ataques por análise de padrão. Para evitar isso, o OAEP introduz aleatoriedade, fazendo com que mensagens iguais produzam cifras diferentes a cada execução.

A implementação de OAEP neste projeto funciona em três etapas:

1. **Hashing e formatação (Data Block - DB):**

   O hash de uma string vazia (SHA3-256(‘’)) é concatenado com uma sequência de zeros e um byte 0x01 separador, seguido da mensagem original. Isso garante que a estrutura da mensagem seja padronizada e que a recuperação posterior da mensagem seja possível.

2. **Geração de máscara:**
   Uma semente aleatória (seed) é gerada. Com base nessa semente, é aplicada uma função de mascaramento sobre o bloco de dados (DB), criando o maskedDB. Depois, o maskedDB é usado para gerar uma nova máscara sobre a semente, gerando o maskedSeed.
3. **Codificação final:**

   Os valores maskedSeed e maskedDB são concatenados junto a um byte zero inicial (para alinhamento), resultando em um bloco codificado único. Esse bloco é convertido para número inteiro para ser cifrado com RSA normalmente.

A decodificação (OAEP decifrar) realiza o processo inverso: desfaz o mascaramento da semente e do bloco de dados com os hashes e retira a mensagem original.

Este processo garante que a cifra seja probabilística (não-determinística), seja possível verificar a integridade da estrutura decodificada, e que o algoritmo RSA fique protegido contra ataques baseados no texto escolhido.

Funcionamento do código:

- **Parte I: geração de chaves e cifra RSA:**

O sistema implementa a geração de chaves RSA com primos de pelo menos 1024 bits, utilizando o teste de primalidade de Miller-Rabin.

Após isso, os valores de n, e e d são calculados com base no produto dos primos e na função totiente de Euler. A função modular inversa é implementada diretamente via o algoritmo de Euclides estendido.

- **Parte II: assinatura:**

O processo de assinatura consiste em aplicar a função de hash SHA-3 (permitida pela especificação) sobre o conteúdo da mensagem, seguida da cifração do hash com a chave privada RSA. O resultado é codificado em BASE64 e armazenado junto da mensagem.

- **Parte III: verificação:**

A verificação realiza o parsing do arquivo, separa o conteúdo da mensagem e a assinatura, decodifica a assinatura e a compara com o hash calculado diretamente da mensagem. Caso coincidam, a assinatura é considerada válida.

- **Implementação:**

As funções *gerar_chaves_rsa*, *gerar_primo_grande* e *miller_rabin* compõem a parte responsável pela criação de chaves RSA seguras. Números primos grandes são gerados por meio do teste probabilístico de Miller-Rabin. Com dois primos distintos p e q, o módulo $n = p \times q$ e o expoente da chave pública $e = 65537$ são utilizados para calcular o expoente da chave privada d, usando o algoritmo de Euclides estendido (*inverso_modular*). O par de chaves gerado é retornado como um dicionário contendo a chave pública e a chave privada.

Para garantir uma camada adicional de segurança ao ciframento RSA, foi implementado um esquema de codificação através das funções *oaep_cifrar* e *oaep_decifrar*. Nelas, a mensagem é estruturada com base em um hash *SHA3-256* vazio, acrescida de um padding (*ps*) e de um separador (*\x01*). Em seguida, aplica-se uma máscara de bits sobre o bloco de dados e sobre a semente aleatória (*seed*), garantindo aleatoriedade e resistência a ataques de texto conhecido. Esse mecanismo serve tanto para cifragem quanto para assinatura segura.

A assinatura digital é realizada pela função *assinar_mensagem*, que aplica a função de hash *SHA3-256* à mensagem original e, em seguida, cifra o digest com a chave privada RSA, gerando a assinatura. A assinatura é codificada em *Base64* para facilitar seu armazenamento e transporte.

A verificação de assinaturas é feita pela função *verificar_assinatura*. Ela decifra a assinatura com a chave pública RSA e compara o hash resultante com o hash real da mensagem recebida. Caso os valores coincidam, a assinatura é considerada válida. Isso assegura a autenticidade da mensagem.

As funções *salvar_arquivo_assinado* e *carregar_arquivo_assinado* são responsáveis por armazenar e recuperar mensagens assinadas em arquivos. O formato utilizado separa o conteúdo da mensagem e a assinatura usando marcadores (*BEGIN/END MESSAGE*, *BEGIN/END SIGNATURE*). A mensagem é armazenada em *Base64* e a assinatura como uma string codificada.

No bloco principal (*if name == "main"*), o script executa a geração das chaves RSA, carrega uma mensagem de um arquivo (*mensagem.txt*), gera a assinatura digital, salva a mensagem assinada em um novo arquivo (*mensagem_assinada.txt*) e, por fim, realiza a verificação da assinatura. O resultado da verificação é impresso na tela, confirmando se a assinatura corresponde à mensagem original.