Trabalho segurança computacional

(Relatório verificador de assinatura)

Luiz Henrique Figueiredo Soares - 211068403

1 de setembro de 2024

professor: João José Costa Gondim

1. Introdução

Este relatório detalha a implementação de um sistema de criptografia utilizando RSA (Rivest–Shamir–Adleman) combinado com o esquema de padding OAEP (Optimal Asymmetric Encryption Padding). A implementação também utiliza a função hash SHA-3 para fortalecer a segurança do esquema de criptografia. A seguir, são abordados os conceitos fundamentais, as cifras e operações implementadas, a descrição detalhada do código-fonte, e um descritivo sobre o funcionamento do RSA, OAEP e assinatura RSA.

2. Descrição das Cifras, Modo de Operação e Operações Implementadas

2.1 RSA (Rivest–Shamir–Adleman)

O RSA é uma cifra assimétrica, o que significa que utiliza um par de chaves: uma chave pública para criptografar dados e uma chave privada para descriptografá-los. No contexto do código implementado, o RSA é utilizado para proteger uma mensagem convertendo-a em um texto cifrado que só pode ser revertido com a chave privada correspondente.

2.2 OAEP (Optimal Asymmetric Encryption Padding)

O OAEP é um esquema de padding que aumenta a segurança do RSA, prevenindo ataques de texto cifrado escolhidos. Ele faz isso combinando a mensagem com uma sequência de bytes aleatórios e aplicando funções hash (neste caso, SHA-3) para garantir que a mesma mensagem criptografada em diferentes instâncias produza diferentes textos cifrados.

2.3 SHA-3

O SHA-3 é uma função de hash criptográfica que gera um valor de hash a partir de uma entrada de dados, criando uma "impressão digital" única. Nesta implementação, o SHA-3 é usado tanto para gerar hashes como parte do OAEP quanto para verificar a integridade dos dados após a descrição.

2.4 Modo de Operação

O código implementa a criptografia RSA no modo de operação de chave pública. A operação ocorre em duas etapas principais:

- Criptografia: A mensagem é cifrada usando a chave pública e o padding OAEP.
- Descriptografar: O texto cifrado é revertido ao seu estado original usando a chave privada.

3. Descrição das Implementações

3.1 Geração das Chaves RSA

No seu código, a geração das chaves RSA segue os passos tradicionais, onde dois números primos grandes (**`p_prime`** e **`q_prime`**) são gerados aleatoriamente. O produto desses números (**n**, o módulo RSA) e a chave pública (**e**) são determinados, e a chave privada (**d**) é calculada como o inverso modular de **e**. O código também inclui o teste de primalidade usando o método Miller-Rabin para garantir que os números gerados sejam primos.

```
# Gera dois números primos grandes de 1024 bits
while True:
    p_prime = os.urandom(128)
    if p_prime not in tested_primes:
        if is_prime(int.from_bytes(p_prime, sys.byteorder), prime_test_iterations):
            print(f"p_prime :\n{int.from_bytes(p_prime, sys.byteorder)}\n")
            while True:
                q_prime = os.urandom(128)
                if p_prime != q_prime:
                    if q_prime not in tested_primes:
                        if is_prime(int.from_bytes(q_prime, sys.byteorder), prime_test_iterations):
                            print(f"q_prime :\n{int.from_bytes(q_prime, sys.byteorder)}\n")
                            break
                        tested_primes.append(q_prime)
            break
        tested_primes.append(p_prime)

# Converte os números primos para inteiros
p_prime = int.from_bytes(p_prime, sys.byteorder)
q_prime = int.from_bytes(q_prime, sys.byteorder)
```

Imagem 01 - Implementação das chaves RSA

A chave privada **d** é calculada utilizando o inverso modular de **e**, garantindo que seja possível descriptografar o texto cifrado com segurança.

```
# Inverso modular de e
d_private_exponent = modular_inverse(e_public_exponent, lcm_phi_n)

print(f'd_private_exponent :\n{d_private_exponent}\n')
```

Imagem 02 - Inverso do modular

3.2 Criptografia com OAEP

O esquema de padding OAEP é aplicado para reforçar a segurança da criptografia RSA. O código combina a mensagem original com uma string de padding e usa SHA-3 para gerar um hash seguro da label (rótulo). O OAEP é um processo que garante que mensagens idênticas não produzam o mesmo texto cifrado.

```python
# Label para o hash do bloco de dados do OAEP
label = "testando".encode()

label_hash = hashlib.sha3_256(label).digest()

hash_length = len(label_hash)

padding_string = generate_padding_string(k_length, message_length, hash_length)

data_block = label_hash + padding_string + b'\x01' + message
```

Imagem 03 - Padding OEAP

O código então codifica a mensagem usando a função **`mgf1`** (Mask Generation Function 1) e a cifra RSA para gerar o texto cifrado:

```python
# Recuperando o hash com SHA3-256
label_hash = hashlib.sha3_256(label).digest()

masked_seed = decrypted_message_bytes[1:hash_length + 1]
masked_data_block = decrypted_message_bytes[hash_length + 1:]

seed_mask = mgf1(masked_data_block, hash_length)

seed = bytes([a ^ b for a, b in zip(masked_seed, seed_mask)])

data_block_mask = mgf1(seed, k_length - hash_length - 1)

data_block = bytes([a ^ b for a, b in zip(masked_data_block, data_block_mask)])

label_hash_verify = data_block[:hash_length]
```

Imagem - 04 codificação com mgf1

3.3 Descriptografia com Verificação de Integridade

A descriptografia no RSA reverte o processo de criptografia, utilizando a chave privada para converter o texto cifrado de volta para a mensagem original. A integridade da mensagem é verificada com o hash gerado durante o padding OAEP, garantindo que a mensagem não foi alterada.

```python
decrypted_integer_message = pow(ciphertext, d_private_exponent, rsa_modulus_n)

# Transforma o inteiro decifrado em bytes
decrypted_message_bytes = decrypted_integer_message.to_bytes(encoded_message_length, byteorder='big')

# Recuperando o hash com SHA3-256
label_hash = hashlib.sha3_256(label).digest()
```

Imagem 05 - descriptografia no RSA

```
218 # Verifica se o Hash é o mesmo no bloco de dados após a decifração
219 if label_hash_verify ≠ label_hash:
220     print('ERRO')
221     sys.exit(1)
222
223 remainder = data_block[hash_length:]
224
225 counter = 0
226
```

Imagem 06 - Verificação do Hash

4. Descritivo do RSA, OAEP, e Assinatura RSA

4.1 RSA

O RSA é um dos sistemas de criptografia assimétrica mais conhecidos, baseado na dificuldade de fatorar grandes números compostos. Ele é amplamente utilizado para garantir a confidencialidade e autenticidade em comunicações digitais.

4.2 OAEP

O OAEP é um padding que torna o RSA mais seguro ao introduzir aleatoriedade no processo de criptografia, garantindo que ataques de texto cifrado escolhidos sejam ineficazes.

4.3 Assinatura RSA

A assinatura RSA utiliza a chave privada para criar uma assinatura digital de um documento ou mensagem, que pode ser verificada com a chave pública correspondente. Embora o código fornecido se concentre na criptografia e descriptografia, a assinatura RSA segue princípios semelhantes à criptografia assimétrica.

Conclusão

A implementação descrita combina a robustez do algoritmo RSA com o padding OAEP, utilizando SHA-3 para garantir a segurança criptográfica. Este relatório abordou o funcionamento das cifras, o modo de operação, as operações implementadas, e apresentou o código fonte completo para referência e utilização.

O código demonstra como aplicar práticas criptográficas modernas para proteger dados sensíveis em sistemas digitais.

Referencias

1 - Informaçoes sobre o SHA-3: https://pt.wikipedia.org/wiki/SHA-3

2 - Docuemntação da biblioteca hashlib: https://docs.python.org/3/library/hashlib.html

3 - Miller-Rabin test: https://en.wikipedia.org/wiki/Miller%E2%80%93Rabin_primality_test

4 - Documentação Numpy: https://numpy.org/doc/