Trabalho 3
Gerador/Verificador de Assinaturas

Davi Jesus de Almeida Paturi, 20/0016784

[1]Dep. Ciência da Computação – Universidade de Brasília (UnB)
CIC0201 - Segurança Computacional

davi.paturi@aluno.unb.br

Resumo. *O RSA (Rivest-Shamir-Adleman) é um dos algoritmos de criptografia assimétrica mais amplamente utilizados. Ele foi inventado em 1977 por Ron Rivest, Adi Shamir e Leonard Adleman. A criptografia assimétrica envolve o uso de um par de chaves, uma chave pública para criptografar dados e uma chave privada correspondente para descriptografar.*

1. Introdução

A criptografia desempenha um papel crítico na segurança da informação no mundo digital. O algoritmo RSA, nomeado após seus criadores Ron Rivest, Adi Shamir e Leonard Adleman, é um dos pilares da criptografia assimétrica. Neste trabalho, exploraremos os princípios básicos do algoritmo de criptografia RSA e a técnica de padding OAEP.

Esse trabalho está dividido em 4 seções. A seção 2 apresenta as ferramentas e os metódos utilizados para o desenvolvimento desse trabalho. Na seção 3 está definição dos modelos de cifragem e decifragem usando o algoritmo RSA. A seção 4 exibe a definição da técnica de padding OAEP. Na seção 5 estão expostos os algoritmos utilizados para a implementação do trabalho.

2. Metodologia

A implementação dos algoritmos de geração de chaves, cifragem RSA, assinatura e verificação foram feitos em Python. O código fonte das implementações estão disponíveis em [Github 2023].

3. RSA

O RSA (Rivest-Shamir-Adleman) é um dos algoritmos de criptografia assimétrica mais amplamente utilizados. Ele foi inventado em 1977 por Ron Rivest, Adi Shamir e Leonard Adleman. A criptografia assimétrica envolve o uso de um par de chaves, uma chave pública para criptografar dados e uma chave privada correspondente para descriptografar.

3.1. Geração de Chaves

A partir de dois primos grandes e distintos, p e q, calcule o módulo n, como $n = pq$. Calcule a função totiente de Euler $\phi(n) = (p-1)(q-1)$.

A chave pública é definida a partir de um número inteiro e que seja co-primo com $\phi(n)$, ou seja, $mdc(e, \phi(n)) = 1$. Geralmente e é escolhido como um número primo pequeno, o mais comum é 65537 ($2^{16} + 1$).

A chave privada é definida a partir de um inteiro d, que deve satisfazer $de \equiv 1 \ (mod \ \phi(n))$. É possível calcular d utilizando o algoritmo extendido de Euclides.

A chave pública final é (n, e) e a chave privada é (n, d).

3.2. Criptografia e Descriptografia

3.2.1. Criptografia

O remetente usa a chave pública (n, e) do destinatário para criptografar a mensagem M. A criptografia é feita usando a operação $C \equiv M^e \ (mod \ n)$.

3.2.2. Descriptografia

O destinatário usa sua chave privada (n, d) para descriptografar a mensagem cifrada C. A operação de descriptografia é $M \equiv C^d \ (mod \ n)$.

4. OAEP

OAEP, ou Optimal Asymmetric Encryption Padding, é uma técnica de padding (preenchimento) utilizada em criptografia assimétrica, especialmente em esquemas de criptografia de chave pública, como RSA. O principal objetivo do OAEP é adicionar uma camada de aleatoriedade aos dados antes da criptografia, proporcionando maior segurança contra ataques conhecidos como ataques de escolha de texto simples (chosen plaintext attacks).

É importante observar que o OAEP é mais comumente associado ao RSA, mas a técnica em si pode ser usada com outros algoritmos de criptografia de chave pública. Quando você estiver implementando criptografia em suas aplicações, certifique-se de seguir as práticas recomendadas e padrões de segurança atualizados.

A função do OAEP é proteger contra determinados tipos de ataques, como ataques de preenchimento (padding attacks) e outros ataques criptográficos. Essa técnica é composta por alguns passos, que serão explicados brevemente a seguir.

4.1. Adição de Aleatoriedade

Antes de criptografar os dados, o OAEP adiciona aleatoriedade aos mesmos. Isso ajuda a evitar que um atacante explore padrões nos dados originais que possam ser refletidos nos dados cifrados. Após gerar uma máscara de aleatoriedade R utilizando uma função hash segura, essa máscara é combinada com os dados originais para adicionar a aleatoriedade. Uma segunda função hash segura é utilizada para gerar uma máscara de redundância G. Esta máscara é combinada com a combinação dos dados originais e a máscara de aleatoriedade.

4.2. Concatenação e Criptografia

As máscaras de aleatoriedade e redundância são concatenadas aos dados originais, formando um bloco maior. Esse bloco é então criptografado utilizando o algoritmo de criptografia assimétrica (por exemplo, RSA). O resultado cifrado é o que é enviado ou armazenado de forma segura.

4.3. Descriptografia

No processo de descriptografia, a operação inversa é realizada para remover a aleatoriedade adicionada durante a criptografia. O OAEP é projetado para fornecer propriedades de segurança específicas e garantir que a criptografia seja resistente a certos tipos de ataques. Ele é frequentemente usado em combinação com esquemas de criptografia assimétrica, especialmente quando é necessário um alto nível de segurança.

5. Código

5.1. Teste de primalidade (Miller-Rabin)

```python
def test_prime(n, witness):
    exp, rem = n - 1, 0
    while not exp & 1:
        exp >>= 1
        rem += 1
    x = pow(witness, exp, n)
    if x == 1 or x == n - 1:
        return True
    for _ in range(rem - 1):
        x = pow(x, 2, n)
        if x == n - 1:
            return True
    return False

def is_prime(n, k = 40):
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0 or n % 3 == 0:
        return False
    for _ in range(k):
        witness = randrange(2, n - 1)
        if not test_prime(n, witness):
            return False
    return True
```

5.2. Geração de chaves

```python
def generate_keys():
    prime_p = generate_prime()
    prime_q = generate_prime()
    modulus = prime_p * prime_q
    totient = (prime_p - 1) * (prime_q - 1)
    encryption_exponent = 65537
    out_mdc = mdc(encryption_exponent, totient)
    decryption_exponent = out_mdc[1]
    if decryption_exponent < 0:
        decryption_exponent += totient
    public_key = [modulus, encryption_exponent]
    private_key = [modulus, decryption_exponent]
    return (public_key, private_key)
```

5.3. Cifração e Decifração RSA

```
def rsa_encrypt(encoded_message, public_key):
    return [pow(i, public_key[1], public_key[0]) for i in
    encoded_message]

def rsa_decrypt(encoded_message, private_key):
    return [pow(i, private_key[1], private_key[0]) for i in
    encoded_message]
```

5.4. Cifração e Decifração OAEP

```
def oaep_encrypt(message, public_key, label=""):
    label = label.encode()
    max_len = K - 2 * H_LENGTH - 2
    if len(message) > max_len:
        raise ValueError(f"Mensagem muito longa para ser codificada
    utilizando OAEP. Tamanho m ximo: {max_len}")
    l_hash = sha256(label).digest()
    db = form_data_block(l_hash, message)
    seed = urandom(H_LENGTH)
    db_mask = mgf1(seed, K - H_LENGTH - 1, sha256)
    masked_db = bytes(xor(db, db_mask))
    seed_mask = mgf1(masked_db, H_LENGTH, sha256)
    masked_seed = bytes(xor(seed, seed_mask))
    encoded_message = b'\x00' + masked_seed + masked_db
    return rsa_encrypt(encoded_message, public_key)

def oaep_decrypt(encoded_message, private_key, label=""):
    label = label.encode()
    encoded_message = rsa_decrypt(list(encoded_message), private_key)
    l_hash = sha256(label).digest()
    if len(encoded_message) != K:
        raise ValueError(f"Tamanho da mensagem codificada    inv lido.
     O tamanho deve ser {K}.")
    masked_seed = bytes(encoded_message[1 : H_LENGTH + 1])
    masked_db = bytes(encoded_message[H_LENGTH + 1:])
    seed_mask = mgf1(masked_db, H_LENGTH, sha256)
    seed = bytes(xor(masked_seed, seed_mask))
    db_mask = mgf1(seed, K - H_LENGTH - 1, sha256)
    db = bytes(xor(masked_db, db_mask))
    l_hash_gen = db[:H_LENGTH]
    if l_hash_gen != l_hash:
        raise ValueError("Hash da mensagem decodificada    inv lida.")
    message_start = H_LENGTH + db[H_LENGTH:].find(b'\x01') + 1
    message = db[message_start:]
    return message
```

Referências

[Github 2023] Github (2023). Projeto 3. https://github.com/davipatury/SC-T3. [Online].