Relatório - RSA

Matheus Guaraci Lima Bouças Alves - 180046951
Aline Mitiko Otsubo - 170004601

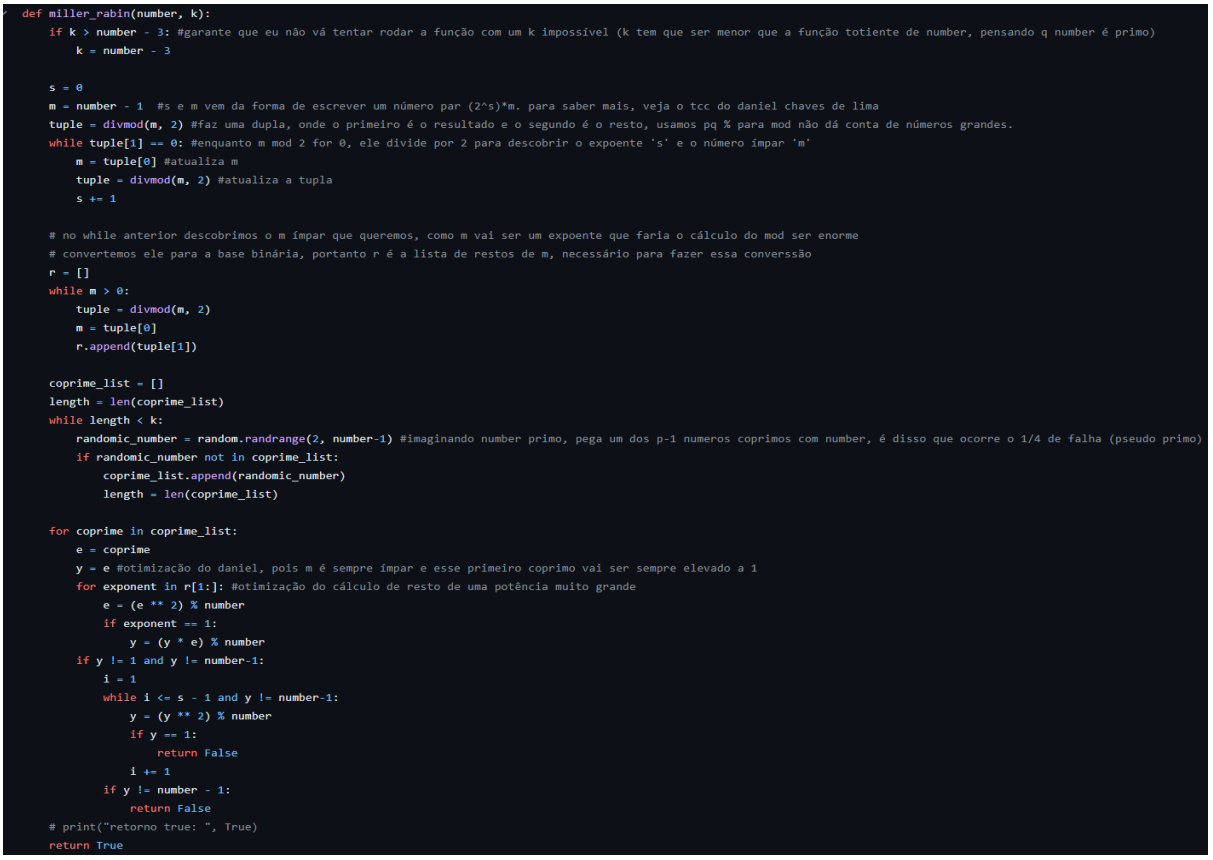
Dep. Ciência da Computação – Universidade de Brasília (UnB)

1. Introdução

Este trabalho elabora a implementação de um gerador e verificador de assinaturas RSA em python. Utilizamos as assinaturas para garantir a autenticidade e integridade das mensagens cifradas com o RSA. No algoritmo RSA, a geração de chaves é feita utilizando números primos grandes. Qualquer pessoa pode fazer a cifragem de uma mensagem utilizando a chave pública, mas apenas o detentor da chave privada consegue decodificá-la.

2. Testando possíveis números primos

Nosso programa sorteia números ímpares de 1024 bits e testa se ele é primo utilizando o teste de primalidade Miller-Rabin, o qual aplica o pequeno teorema de Fermat assumindo que o número sorteado seja primo. O teste de Miller possui alguns casos em que há incerteza sobre a primalidade do número, mas Rabin mostrou que os casos inconclusivos possuem uma probabilidade baixa de não serem primos. A imagem a seguir mostra a implementação do teste.



```python
def miller_rabin(number, k):
    if k > number - 3: #garante que eu não vá tentar rodar a função com um k impossível (k tem que ser menor que a função totiente de number, pensando q number é primo)
        k = number - 3

    s = 0
    m = number - 1  #s e m vem da forma de escrever um número par (2^s)*m. para saber mais, veja o tcc do daniel chaves de lima
    tuple = divmod(m, 2) #faz uma dupla, onde o primeiro é o resultado e o segundo é o resto, usamos pq % para mod não dá conta de números grandes.
    while tuple[1] == 0: #enquanto m mod 2 for 0, ele divide por 2 para descobrir o expoente 's' e o número ímpar 'm'
        m = tuple[0] #atualiza m
        tuple = divmod(m, 2) #atualiza a tupla
        s += 1

    # no while anterior descobrimos o m ímpar que queremos, como m vai ser um expoente que faria o cálculo do mod ser enorme
    # convertemos ele para a base binária, portanto r é a lista de restos de m, necessário para fazer essa converssão
    r = []
    while m > 0:
        tuple = divmod(m, 2)
        m = tuple[0]
        r.append(tuple[1])

    coprime_list = []
    length = len(coprime_list)
    while length < k:
        randomic_number = random.randrange(2, number-1) #imaginando number primo, pega um dos p-1 numeros coprimos com number, é disso que ocorre o 1/4 de falha (pseudo primo)
        if randomic_number not in coprime_list:
            coprime_list.append(randomic_number)
            length = len(coprime_list)

    for coprime in coprime_list:
        e = coprime
        y = e #otimização do daniel, pois m é sempre ímpar e esse primeiro coprimo vai ser sempre elevado a 1
        for exponent in r[1:]: #otimização do cálculo de resto de uma potência muito grande
            e = (e ** 2) % number
            if exponent == 1:
                y = (y * e) % number
        if y != 1 and y != number-1:
            i = 1
            while i <= s - 1 and y != number-1:
                y = (y ** 2) % number
                if y == 1:
                    return False
                i += 1
            if y != number - 1:
                return False
    # print("retorno true: ", True)
    return True
```

Uma vez que temos uma boa chance do número escolhido ser primos, utilizamos a função "gen_keys_pair" para fazermos a geração do par de chaves pública e privada.

3. Cifragem e decifragem

Para cifrar uma mensagem com o RSA de forma segura, utilizamos o OAEP, o qual preenche a mensagem e faz ela passar duas vezes pela cifra de Feistel. Como podemos observar no código, a mensagem recebida é preenchida com zeros, passa pela malha sendo mascarada com o hash sha1 e então é retornada.

```
def oaep_encode(b_message, k, label=b''):
    db = sha1(label) + b'\x00' * (k - len(b_message) - 2 * len(sha1(label)) - 2) + b'\x01' + b_message
    seed = os.urandom(len(sha1(label)))
    db_mask = mgf1(seed, k - len(sha1(label)) - 1)
    masked_db = xor(db, db_mask)
    seed_mask = mgf1(masked_db, len(sha1(label)))
    masked_seed = xor(seed, seed_mask)
    return b'\x00' + masked_seed + masked_db
```

A partir da mensagem mascarada do OAEP, a função "encrypt_oaep" consegue realizar a cifragem da mensagem chamando funções auxiliares, para que o retorno do OAEP seja transformando num inteiro, calculado o módulo para a cifração, e só então retornar a cifragem numa string em bytes.

Para recuperarmos a mensagem, chamamos a função "decrypt_oaep", tendo como argumentos o texto cifrado e a chave privada, nela é chamada a função "oaep_decode", a qual remove o preenchimento e recupera a mensagem, desfazendo o processo que foi feito na cifra de Feistel da função "oaep_encode".

4. Assinatura e verificação

A função "signature" é a responsável por realizar a assinatura usando o RSA. Primeiramente, é calculado o valor hash SHA-3 da mensagem, em sequência, a mensagem é cifrada com RSA. Enquanto a função "verify_signature" calcula o hash da mensagem original e compara com o hash gerado pela mensagem decifrada.

```
def signature(message, public_key):
    hashed = sha3_256(message.encode('utf-8')).digest()
    signature = encrypt_rsa(int.from_bytes(hashed, "big"), public_key)
    signature_bytes = signature.to_bytes((signature.bit_length() + 7) // 8, 'big')
    return encode_base64(signature_bytes)

def verify_signature(signature, message, private_key):
    signature = decode_base64(signature)
    hashed = sha3_256(message).digest()
    if decrypt_rsa(int.from_bytes(signature, "big"), private_key) == int.from_bytes(hashed, "big"):
        print("assinatura válida :)")
    else:
        print("assinatura inválida :(")
```

5. Referências

LIMA, Daniel Chaves. **Algoritmo para o teste de primalidade Miller-Rabin**. Trabalho de conclusão de curso (graduação) - Faculdade de Matemática, Universidade Federal do Pará. Pará, 2019.

KATZ, Jonathan; LINDELL, Yehuda. Introduction to modern cryptography. 2.ed. Boca Raton: Chapman & Hall/CRC, 2014.

CONTEÚDO aberto. In: WIKIPÉDIA: a enciclopédia livre. Disponível em: https://pt.wikipedia.org/wiki/RSA_(sistema_criptogr%C3%A1fico)