Lista de Exercícios 2: Algoritmo RSA

Giulia Moura Ferreira, 20/00018795

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

giulia.ferreira@aluno.unb.br

1. Introdução

Este relatório tem como objetivo apresentar a resolução detalhada da Lista de Exercícios 02 da disciplina de Segurança Computacional. Os exercícios propostos exploram o funcionamento do algoritmo RSA, abordando aspectos teóricos e práticos de criptografia assimétrica. Cada questão foi desenvolvida com o uso de Python para realizar os cálculos necessários e demonstrar os processos de encriptação e decriptação em diferentes cenários.

O algoritmo RSA (Rivest-Shamir-Adleman) é um dos métodos mais conhecidos e amplamente utilizados na criptografia assimétrica. Ele permite a troca segura de informações através de chaves pública e privada, garantindo a confidencialidade e integridade dos dados. [3]

O repositório do projeto pode ser encontrado a partir desse link.

2. Desenvolvimento

2.1. Atividade 1: Implementação do Algoritmo RSA para Encriptação e Decriptação

Este exercício teve como objetivo implementar o RSA para realizar encriptação e decriptação de mensagens numéricas, explorando conceitos fundamentais como cálculo de n, função totiente $\phi(n)$, e o inverso modular.

Neste contexto, foram realizados os seguintes passos para cada subquestão [2]:

1. Calcular os valores de n (produto de dois números primos p e q) e $\phi(n)$ (função totiente de Euler)
2. Verificar se o valor e escolhido é válido (coprimo com $\phi(n)$)
3. Determinar o valor de d, que é o inverso modular de e em relação a $\phi(n)$
4. Realizar a encriptação da mensagem M utilizando $C = M^e mod n$
5. Realizar a decriptação da mensagem cifrada C utilizando $M = C^e mod n$

A implementação foi realizada em Python, de forma a automatizar os cálculos e garantir a precisão dos resultados.

```
from sympy import mod_inverse
import pandas as pd
```

Figure 1. Bibliotecas importadas

Essa parte importa as funções necessárias:

- **mod_inverse**: Para calcular o inverso modular de e em relação a $\phi(n)$. **Essa biblioteca foi importada em todas as atividades dessa lista**
- **pandas**: Para criar tabelas organizadas com os resultados dos cálculos RSA em cada questão da lista

```python
def rsa_encrypt_decrypt(p, q, e, M):  1 usage
    n = p * q
    phi_n = (p - 1) * (q - 1)

    d = mod_inverse(e, phi_n)
    C = pow(M, e, n)
    M_decrypted = pow(C, d, n)

    return {
        "n": n,
        "ϕ(n)": phi_n,
        "e": e,
        "d": d,
        "Encriptado (C)": C,
        "Decriptado (M)": M_decrypted
    }
```

Figure 2. Função *rsa_encrypt_decrypt*

Essa parte define a função que realiza os cálculos de encriptação e decriptação com os parâmetros p, q, e e M, segue a explicação de cada parte apresentada:

1. Calcula n como o produto dos números primos p e q
2. Calcula $\phi(n) = (p-1)(q-1)$ (*phi_n*), essencial para encontrar a chave privada d
3. Calcula d, o inverso modular de e em relação a $\phi(n)$. d será usado na decriptação
4. Realiza a encriptação utilizando $C = M^e \ mod \ n$
5. Realiza a decriptação utilizando $M = C^d \ mod \ n$
6. Por fim, retorna os valores calculados: n, $\phi(n)$, e, d, o texto cifrado C e a mensagem decriptografada M.

a. $p = 3; q = 11, e = 7; M = 5$

b. $p = 5; q = 11, e = 3; M = 9$

c. $p = 7; q = 11, e = 17; M = 8$

d. $p = 11; q = 13, e = 11; M = 7$

e. $p = 17; q = 31, e = 7; M = 2$

```python
exercises = [
    {"p": 3, "q": 11, "e": 7, "M": 5},
    {"p": 5, "q": 11, "e": 3, "M": 9},
    {"p": 7, "q": 11, "e": 17, "M": 8},
    {"p": 11, "q": 13, "e": 11, "M": 7},
    {"p": 17, "q": 31, "e": 7, "M": 2}
]
```

Figure 3. Testes propostos na lista

```
results = []
for idx, ex in enumerate(exercises, start=1):
    result = rsa_encrypt_decrypt(ex["p"], ex["q"], ex["e"], ex["M"])
    result["Exercício"] = f"(1-{chr(96 + idx)})"
    results.append(result)

results_df = pd.DataFrame(results)
print(results_df)
```

Figure 4. Processamento da função *rsa_encrypt_decrypt*

Essa parte processa os exercícios de RSA, armazena os resultados em uma lista e organiza tudo em um DataFrame para exibição tabular:

n	ϕ(n)	e	d	Encriptado (C)	Decriptado (M)	Exercício
33	20	7	3	14	5	(1-a)
55	40	3	27	14	9	(1-b)
77	60	17	53	57	8	(1-c)
143	120	11	11	106	7	(1-d)
527	480	7	343	128	2	(1-e)

Figure 5. Output da Atividade 1

2.2. Atividade 2: Recuperação de Mensagem Interceptada

Neste exercício, o objetivo é decriptografar uma mensagem interceptada C, enviada utilizando o algoritmo RSA. O desafio consiste em recuperar a mensagem original M, mesmo sem acesso direto aos valores primos p e q utilizados para gerar n [2]. Para isso, é necessário:

1. Fatorar n para determinar p e q
2. Calcular $\phi(n)$
3. Encontrar d
4. Utilizar d para decriptografar C e recuperar M

```python
def rsa_decrypt_intercepted(C, e, n):  1 usage
    for i in range(2, n):
        if n % i == 0:
            p = i
            q = n // i
            break

    phi_n = (p - 1) * (q - 1)
    d = mod_inverse(e, phi_n)
    M = pow(C, d, n)

    return {
        "n": n,
        "ϕ(n)": phi_n,
        "e": e,
        "d": d,
        "Mensagem decriptada (M)": M
    }
```

Figure 6. Função *rsa_decrypt_intercepted*

Essa parte define a função responsável por decriptografar a mensagem interceptada, utilizando os parâmetros C (mensagem cifrada), e (chave pública) e n (produto dos primos):

1. O loop identifica os dois fatores primos p e q de n, assumindo que n é pequeno e fatorável diretamente
2. Após encontrar p e q, calcula $\phi(n)$ como $(p-1)(q-1)$, necessário para encontrar d
3. Calcula d, o inverso modular de e em relação a $\phi(n)$, usado na decriptação da mensagem
4. Decriptografa a mensagem cifrada C utilizando $M = C^d \ mod \ n$, recuperando, assim, a mensagem original M
5. Retorna um dicionário contendo os valores calculados, que incluem n, $\phi(n)$, e, d e a mensagem decriptografada M

```python
C, e, n = 10, 5, 35
for item, result in rsa_decrypt_intercepted(C, e, n).items():
    print(f'{item} = {result}')
```

Figure 7. Processamento da função *rsa_decrypt_intercepted*

Essa parte processa a função com os parâmetros fornecidos no enunciado da atividade, sendo o texto cifrado $C = 10$ enviado a um usuário cuja a chave pública é $e = 5$, $n = 35$. Obtendo o seguinte resultado, quando executada:

```
n = 35
ϕ(n) = 24
e = 5
d = 5
Mensagem decriptada (M) = 5
```

Figure 8. Output da Atividade 2

Essa implementação demonstra a vulnerabilidade do RSA quando n n é pequeno, reforçando a necessidade de números primos grandes para garantir a segurança do algoritmo.

2.3. Atividade 3: Encriptação e Decriptação de Mensagem Binária

Na terceira atividade da lista, o objetivo é aplicar o algoritmo RSA para encriptar e decriptar uma mensagem binária (m) representada como uma sequência de bits (0s e 1s). Diferentemente do processamento bit a bit, aqui a mensagem binária é tratada como um número decimal para realizar a encriptação e a decriptação. [1]. A solução envolve os seguintes passos:

1. Converter a mensagem binária em um número decimal
2. Determinar d, o inverso modular de e em relação a $\phi(n)$
3. Encriptar a mensagem decimal usando $C = M^e \ mod \ n$
4. Decriptar a mensagem cifrada usando $M = C^d \ mod \ n$
5. Converter o número decimal decriptado de volta para binário

```python
def rsa_encrypt_decrypt_binary(p, q, e, binary_message):  1 usage
    decimal_message = int(binary_message, 2)

    n = p * q
    phi_n = (p - 1) * (q - 1)
    d = mod_inverse(e, phi_n)

    encrypted_decimal = pow(decimal_message, e, n)
    decrypted_decimal = pow(encrypted_decimal, d, n)

    encrypted_binary = bin(encrypted_decimal)[2:]
    decrypted_binary = bin(decrypted_decimal)[2:].zfill(len(binary_message))

    return {
        "n": n,
        "ϕ(n)": phi_n,
        "e": e,
        "d": d,
        "Mensagem original (binário)": binary_message,
        "Mensagem original (decimal)": decimal_message,
        "Mensagem encriptada (binário)": encrypted_binary,
        "Mensagem encriptada (decimal)": encrypted_decimal,
        "Mensagem decriptada (binário)": decrypted_binary,
        "Mensagem decriptada (decimal)": decrypted_decimal
    }
```

Figure 9. Função *rsa_encrypt_decrypt_binary*

Essa parte define a função para encriptar e decriptar a mensagem binária usando RSA, tendo como parâmetro os números primos p e q, a chave pública e e a mensagem binária em forma de string $binary\_message$:

1. Converte a mensagem binária m em um número decimal
2. Calcula n, o produto dos números primos p e q
3. Calcula $\phi(n)$
4. Calcula d, o inverso modular de e em relação a $\phi(n)$
5. Aplica o RSA na mensagem decimal para fazer a encriptação: $C = M^e \ mod \ n$
6. Aplica o RSA na mensagem decimal cifrada para fazer a decriptação: $M = C^e \ mod \ n$
7. Converte os números decimais decriptados e encriptados de volta para binário, retirando o prefixo "0b" da representação binária
8. Retorna os valores calculados, incluindo as mensagens decimais e binárias, encriptadas e decriptadas

```
p, q, e = 11, 23, 3
binary_message = "0111001"
result_ex3 = rsa_encrypt_decrypt_binary(p, q, e, binary_message)

for item, result in result_ex3.items():
    print(f'{item} = {result}')
```

Figure 10. Processamento da função *rsa_encrypt_decrypt_binary*

Aqui, a função é processada com os parâmetros fornecidos no enunciado da atividade, sendo a mensagem binária $m = 0111001$, $p = 11$, $q = 23$ e $e = 3$. Após o processamento da função, a seguinte saída é obtida:

```
n = 253
ϕ(n) = 220
e = 3
d = 147
Mensagem original (binário) = 0111001
Mensagem original (decimal) = 57
Mensagem encriptada (binário) = 11111010
Mensagem encriptada (decimal) = 250
Mensagem decriptada (binário) = 0111001
Mensagem decriptada (decimal) = 57
```

Figure 11. Output da Atividade 3

2.4. Atividade 4: Encriptação e Decriptação de Mensagem em Texto

Nesta atividade, o algoritmo RSA foi aplicado para encriptar e decriptar uma mensagem em texto utilizando a codificação ASCII. Cada caractere do texto é convertido para seu

valor numérico ASCII, permitindo que seja processado pelo RSA. Os passos para resolver essa atividade são:

1. Converter cada caractere da mensagem para seu valor numérico ASCII
2. Calcular $n = p \cdot q$ e $\phi(n)$
3. Determinar d
4. Encriptar cada caractere ASCII usando $C = M^e \ mod \ n$
5. Decriptar cada caractere cifrado usando $M = C^d \ mod \ n$
6. Converter os valores ASCII de volta para caracteres para reconstruir a mensagem original

```python
def rsa_encrypt_decrypt_ascii(p, q, e, message):  1 usage
    n = p * q
    phi_n = (p - 1) * (q - 1)
    d = mod_inverse(e, phi_n)

    ascii_values = [ord(char) for char in message]

    encrypted_ascii = [pow(m, e, n) for m in ascii_values]
    encrypted_message = ''.join(chr(c) for c in encrypted_ascii)

    decrypted_ascii = [pow(c, d, n) for c in encrypted_ascii]
    decrypted_message = ''.join(chr(m) for m in decrypted_ascii)

    return {
        "n": n,
        "ϕ(n)": phi_n,
        "e": e,
        "d": d,
        "Mensagem original": message,
        "Valores encriptados": encrypted_ascii,
        "Mensagem encriptada": encrypted_message,
        "Valores decriptados": decrypted_ascii,
        "Mensagem decriptada": decrypted_message
    }
```

Figure 12. Função *rsa_encrypt_decrypt_ascii*

Essa parte define a função que realiza os cálculos de encriptação e decriptação RSA de uma mensagem, assim como a conversão de ascii em texto e vice-versa:

1. Calcula n, $\phi(n)$ e d
2. Converte cada caractere da mensagem para seu valor numérico ASCII
3. Aplica a fórmula $C = M^e \ mod \ n$ para encriptar cada valor ASCII

4. Converte os valores ascii cifrados para caracteres
5. Aplica a fórmula $M = C^d \ mod \ n$ para decriptar cada valor ASCII cifrado
6. Converte os valores ascii decriptados de volta para caracteres, reconstruindo a mensagem original
7. Retorna um dicionário com os valores calculados, incluindo os valores ascii e as mensagens em formato de texto, encriptados e decriptados

```
p, q, e = 11, 17, 7
message = "HELLO"

result_ex4 = rsa_encrypt_decrypt_ascii(p, q, e, message)
for item, result in result_ex4.items():
    print(f'{item} = {result}')
```

Figure 13. Processamento da função *rsa_encrypt_decrypt_ascii*

A função é processada com os parâmetros fornecidos no enunciado da atividade, sendo eles a mensagem "HELLO", os números primos $p = 11$ e $q = 17$, e a chave pública $e = 7$, a saída obtida é:

```
n = 187
ϕ(n) = 160
e = 7
d = 23
Mensagem original = HELLO
Valores encriptados = [30, 86, 32, 32, 139]
Mensagem encriptada = ␀V  ␀
Valores decriptados = [72, 69, 76, 76, 79]
Mensagem decriptada = HELLO
```

Figure 14. Output da Atividade 3

3. Conclusão

A lista de exercícios 2 permitiu explorar, de maneira prática, a aplicação do algoritmo RSA em diferentes cenários, reforçando os conceitos essenciais da criptografia assimétrica. Nos exercícios, foi abordado:

- Encriptação e decriptação de números inteiros, destacando o cálculo de n, $\phi(n)$ e o uso das chaves e e d
- Recuperação de mensgens interceptadas por meio da fatoração de n, ilustrando vulnerabilidades do RSA com números pequenos
- Encriptação e decriptação de mensagens binárias, demonstrando a flexibilidade do RSA para diferentes formatos de dados

- Aplicação do RSA em textos utilizando a codificação ASCII, destacando a integração do algoritmo com dados textuais

Esses exercícios reforçam a importância de boas práticas no uso do RSA, como a escolha de números primos grandes para garantir segurança e a compreensão das etapas envolvidas no processo.

References

[1] Lucas Santos. Criptografia 1 - criptografia assimétrica com rsa. https://blog.lsantos.dev/criptografia-assimetrica-com-rsa/. Accessed: 2025-01-25.

[2] Rafael Sousa. Entendendo algoritmo rsa (de verdade). https://hackingnaweb.com/criptografia/entendendo-algoritmo-rsa-de-verdade/. Accessed: 2025-01-24.

[3] Equipe TOTVS. O que significa rsa e qual sua relação com criptografia? https://www.totvs.com/blog/gestao-para-assinatura-de-documentos/rsa/. Accessed: 2025-01-24.