

Implementación de RSA con Square and Multiply y Teorema del Resto Chino

Benjamín Alonso Fernández Andrade

Departamento de Ciencias de la Computación e Informática

Universidad de La Frontera (UFRO)

b.fernandez07@ufromail.cl

<https://github.com/Mizuhar4/RSA-Python-algorithm.git>

Temuco, Chile

Abstract—Este documento presenta una implementación del método RSA utilizando el algoritmo de *square and multiply* y el Teorema del Resto Chino para optimizar las operaciones modulares. Se describen las funciones clave usadas en la implementación, incluyendo la generación de claves, el cifrado y descifrado de mensajes, y el uso de técnicas matemáticas como el máximo común divisor y el algoritmo de Euclides extendido.

I. INTRODUCCIÓN

RSA es uno de los algoritmos de cifrado más utilizados en criptografía. Este trabajo incluye la implementación de las principales funciones necesarias para ejecutar el cifrado y descifrado usando RSA, aprovechando optimizaciones como *square and multiply* y el Teorema del Resto Chino (CRT) [1]. Se describen las funciones clave y su propósito en el contexto del algoritmo.

II. FUNCIONES Y EXPLICACIONES

A. Función *mcd*

```
def mcd(a, b):  
    while b != 0:  
        a, b = b, a % b  
    return a
```

Fig. 1: Diagrama de la función *mcd*.

Descripción: La función *mcd* (Máximo Común Divisor) se usa para encontrar el mayor número que divide a dos enteros sin dejar residuo. Es útil en RSA para verificar si dos números son coprimos, especialmente al elegir el exponente e [2].

B. Función *euclides_extendido*

Descripción: La función *euclides_extendido* implementa el algoritmo de Euclides extendido para encontrar el inverso multiplicativo de un número, que es crucial para calcular la clave privada d en RSA [3].

```
def euclides_extendido(a, b):  
    if b == 0:  
        return a, 1, 0  
    gcd, x1, y1 = euclides_extendido(b, a % b)  
    x = y1  
    y = x1 - (a // b) * y1  
    return gcd, x, y
```

Fig. 2: Diagrama de la función *euclides_extendido*.

```
def es_primo(n, k=40):  
    if n <= 1:  
        return False  
    if n == 2:  
        return True  
    if n % 2 == 0:  
        return False  
  
    r, s = 0, n - 1  
    while s % 2 == 0:  
        r += 1  
        s //= 2  
  
    for _ in range(k):  
        a = random.randint(2, n - 1)  
        x = pow(a, s, n)  
        if x == 1 or x == n - 1:  
            continue  
        for _ in range(r - 1):  
            x = pow(x, 2, n)  
            if x == n - 1:  
                break  
        else:  
            return False  
    return True
```

Fig. 3: Diagrama de la función *es_primo*.

C. Función *es_primo*

Descripción: La función *es_primo* determina si un número es primo, lo cual es esencial en RSA para la elección de los números p y q .

D. Función `generar_primo`

```
def generar_primo(bits):
    while True:
        num = random.getrandbits(bits)
        if es_primo(num):
            return num
```

Fig. 4: Diagrama de la función `generar_primo`.

Descripción: La función `generar_primo` genera un número primo aleatorio de una cantidad específica de bits. Es usada para encontrar p y q en el algoritmo RSA.

E. Función `generar_claves`

```
def generar_claves(bits=512):
    p = generar_primo(bits // 2)
    q = generar_primo(bits // 2)
    n = p * q
    phi_n = (p - 1) * (q - 1)

    e = 65537
    if mcd(e, phi_n) != 1:
        raise ValueError("e no es coprimo con  $\phi(n)$ ")

    _, d, _ = euclides_extendido(e, phi_n)
    d = d % phi_n
    if d < 0:
        d += phi_n

    return (e, n), (d, n), (p, q)
```

Fig. 5: Diagrama de la función `generar_claves`.

Descripción: La función `generar_claves` genera las claves públicas y privadas para RSA, usando números primos p y q , y el cálculo del inverso modular para la clave privada.

F. Función `exponenciacion_rapida`

```
def exponenciacion_rapida(base, exp, mod):
    result = 1
    base = base % mod
    while exp > 0:
        if exp % 2 == 1:
            result = (result * base) % mod
        exp = exp >> 1
        base = (base * base) % mod
    return result
```

Fig. 6: Diagrama de la función `exponenciacion_rapida`.

Descripción: La función `exponenciacion_rapida` implementa el algoritmo de *square and multiply*, optimizando el cálculo de potencias modulares, esencial tanto en cifrado como descifrado.

G. Función `cifrar_numero`

```
def cifrar_numero(mensaje_num, clave_publica):
    e, n = clave_publica
    return exponenciacion_rapida(mensaje_num, e, n)
```

Fig. 7: Diagrama de la función `cifrar_numero`.

Descripción: La función `cifrar_numero` cifra un número usando la clave pública RSA mediante la exponenciación rápida.

H. Función `descifrar_numero`

```
def descifrar_numero(mensaje_cifrado, clave_privada):
    d, n = clave_privada
    return exponenciacion_rapida(mensaje_cifrado, d, n)
```

Fig. 8: Diagrama de la función `descifrar_numero`.

Descripción: La función `descifrar_numero` descifra un número utilizando la clave privada RSA y la exponenciación rápida.

I. Función `descifrar_crt`

```
def descifrar_crt(cifrado, clave_privada, p, q):
    d, n = clave_privada
    dp = d % (p - 1)
    dq = d % (q - 1)
    q_inv = euclides_extendido(q, p)[1] % p

    m1 = exponenciacion_rapida(cifrado, dp, p)
    m2 = exponenciacion_rapida(cifrado, dq, q)

    h = (q_inv * (m1 - m2)) % p
    return (m2 + h * q) % n
```

Fig. 9: Diagrama de la función `descifrar_crt`.

Descripción: La función `descifrar_crt` implementa el Teorema del Resto Chino (CRT) para acelerar el descifrado modular en RSA utilizando los factores primos p y q .

```
def mensaje_a_numeros(mensaje):
    return [ord(c) for c in mensaje]
```

Fig. 10: Diagrama de la función `mensaje_a_numeros`.

J. Función `mensaje_a_numeros`

Descripción: La función `mensaje_a_numeros` convierte un mensaje de texto en una secuencia de números, donde cada número es el código ASCII de cada carácter.

K. Función `numeros_a_mensaje`

```
def numeros_a_mensaje(numeros):
    return ''.join([chr(n) for n in numeros])
```

Fig. 11: Diagrama de la función `numeros_a_mensaje`.

Descripción: La función `numeros_a_mensaje` convierte una secuencia de números en su representación original de caracteres, revirtiendo la conversión a ASCII.

L. Función `cifrar_mensaje`

```
def cifrar_mensaje(mensaje, clave_publica):
    bloques = mensaje_a_numeros(mensaje)
    bloques_cifrados = [cifrar_numero(bloque, clave_publica)
    for bloque in bloques]
    return bloques_cifrados
```

Fig. 12: Diagrama de la función `cifrar_mensaje`.

Descripción: La función `cifrar_mensaje` cifra cada carácter de un mensaje convirtiéndolo primero a un número (ASCII), y luego cifrándolo con RSA.

M. Función `descifrar_mensaje`

Descripción: La función `descifrar_mensaje` descifra un mensaje cifrado por RSA, convirtiendo cada número descifrado de vuelta a caracteres.

```
def descifrar_mensaje(bloques_cifrados, clave_privada, p=None,
q=None):
    if p and q:
        bloques_descifrados = [descifrar_crt(bloque,
clave_privada, p, q) for bloque in bloques_cifrados]
    else:
        bloques_descifrados = [descifrar_numero(bloque,
clave_privada) for bloque in bloques_cifrados]
    return numeros_a_mensaje(bloques_descifrados)
```

Fig. 13: Diagrama de la función `descifrar_mensaje`.

REFERENCES

- [1] Neo LCC UMA, "RSA: Presentación," *Tutorial Criptografía*, 2003. [Online]. Available: <https://neo.lcc.uma.es/evirtual/cdd/tutorial/presentacion/rsa.html>. [Accessed: Sept. 21, 2024].
- [2] J. D. Cook, "RSA exponent," *John D. Cook Consulting*, Dec. 12, 2018. [Online]. Available: <https://www.johndcook.com/blog/2018/12/12/rsa-exponent/>. [Accessed: Sept. 21, 2024].
- [3] Universidad Autónoma de Guerrero, "Implementación del Teorema del Resto Chino," *Tesis Maestría*, 2015. [Online]. Available: http://ri.uagro.mx/bitstream/handle/uagro/776/OK15158773_maestria.pdf?sequence=1&isAllowed=y. [Accessed: Sept. 21, 2024].
- [4] "RSA Encryption Algorithm," *YouTube*, Jul. 15, 2020. [Online]. Available: <https://www.youtube.com/watch?v=XOz0NWxIakQ&t=286s>. [Accessed: Sept. 21, 2024].
- [5] "Teorema del Resto Chino y RSA," *YouTube*, Mar. 12, 2022. [Online]. Available: <https://www.youtube.com/watch?v=AjaMZddJIK0>. [Accessed: Sept. 21, 2024].
- [6] Veritas, "What is RSA encryption," *Veritas Technologies LLC*, 2024. [Online]. Available: <https://www.veritas.com/es/mx/information-center/rsa-encryption>. [Accessed: Sept. 21, 2024].
- [7] Python Diario, "Criptografía en Python: RSA," *Python Diario*, Jul. 2020. [Online]. Available: <https://pythondiario.com/2020/07/criptografia-en-python-rsa.html>. [Accessed: Sept. 21, 2024].