Implementación de un Sistema de Cifrado RSA con Cambio de Bases y Teorema del Resto Chino

[Arturo Alonso Avalos Pascual]
[21482639922]
[https://github.com/oruava/CifradoRSA.git]

***Resumen*—Este informe presenta la implementación de un sistema de criptografía de clave pública/privada RSA, utilizando algoritmos de cambio de bases, cuadrado y multiplicación, y el Teorema del Resto Chino para optimizar las operaciones modulares. Se muestran los procesos de generación de claves, cifrado y descifrado de mensajes (describiendo las funciones principales utilizadas), así como las optimizaciones aplicadas para mejorar la eficiencia del sistema.**

I. Introducción

El algoritmo RSA, nombrado por sus creadores Rivest, Shamir y Adleman, es uno de los sistemas de criptografía de clave pública más utilizados en la actualidad. Este informe detalla la implementación de RSA, incluyendo optimizaciones clave para mejorar su rendimiento.

II. Generación de Claves RSA

La generación de claves RSA implica la selección de dos números primos grandes y el cálculo de varios parámetros clave.

II-A. Test de Primalidad de Miller-Rabin

Para generar números primos grandes, se utiliza el test de primalidad de Miller-Rabin:

```
def es_primo(n, k=5):
    if n <= 1 or n == 4:
        return False
    if n <= 3:
        return True

    d = n - 1
    s = 0
    while d % 2 == 0:
        d //= 2
        s += 1

    for _ in range(k):
        a = random.randint(2, n - 2)
        x = pow(a, d, n)
        if x == 1 or x == n - 1:
            continue
        for _ in range(s - 1):
            x = pow(x, 2, n)
            if x == n - 1:
                break
        else:
            return False
    return True
```

Esta función implementa el test de Miller-Rabin, que es un test de primalidad probabilístico. El parámetro k determina el número de iteraciones del test, aumentando la precisión a medida que k aumenta.

II-B. Generación de Claves

Las claves se generan utilizando la siguiente función:

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

    e = 65537
    d = inverso_modular(e, phi)

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

Esta función genera dos números primos p y q, calcula $n = p \times q$ y $\phi(n) = (p-1)(q-1)$, y selecciona e y calcula d como el inverso multiplicativo de e módulo $\phi(n)$.

III. Algoritmo de Cambio de Bases

El cambio de bases se implementa para convertir entre representaciones numéricas:

```
def cambiar_base(num, base):
    if num == 0:
        return [0]
    digitos = []
    while num:
        digitos.append(num % base)
        num //= base
    return digitos[::-1]
```

Esta función convierte un número de base decimal a cualquier base especificada.

IV. Algoritmo de Cuadrado y Multiplicación

La exponenciación modular rápida se implementa mediante el algoritmo de cuadrado y multiplicación:

```
def cuadrado_y_multiplicacion(base,
    exponente, modulo):
    resultado = 1
    base = base % modulo
    while exponente > 0:
        if exponente % 2 == 1:
            resultado = (resultado * base) %
                modulo
```

```
        exponente = exponente >> 1
        base = (base * base) % modulo
    return resultado
```

Este algoritmo reduce significativamente el número de multiplicaciones necesarias para calcular $base^{exponente}$ mód $modulo$.

V. Cifrado y Descifrado

El proceso de cifrado utiliza la clave pública:

```
def cifrar(mensaje, clave_publica):
    e, n = clave_publica
    mensaje_int =
        int.from_bytes(mensaje.encode(), 'big')
    return
        cuadrado_y_multiplicacion(mensaje_int,
        e, n)
```

El descifrado utiliza la clave privada y el Teorema del Resto Chino para optimización:

```
def descifrar(texto_cifrado, clave_privada,
    p, q):
    d, n = clave_privada

    dp = d % (p - 1)
    dq = d % (q - 1)
    qinv = inverso_modular(q, p)

    m1 =
        cuadrado_y_multiplicacion(texto_cifrado
        % p, dp, p)
    m2 =
        cuadrado_y_multiplicacion(texto_cifrado
        % q, dq, q)
    h = (qinv * (m1 - m2)) % p
    m = m2 + h * q

    return m.to_bytes((m.bit_length() + 7) //
        8, 'big').decode()
```

VI. Optimización con el Teorema del Resto Chino

El Teorema del Resto Chino se utiliza en la función de descifrado para acelerar los cálculos. Este método permite realizar dos exponenciaciones más pequeñas (mod p y mod q) en lugar de una exponenciación grande mod n, lo que resulta en una mejora significativa del rendimiento.

VII. Resultados

A continuación, se muestra un ejemplo práctico de la ejecución del sistema de cifrado y descifrado RSA utilizando el código presentado. Para este ejemplo, se generaron claves RSA de 256 bits, se introdujo el mensaje “tomate” y se realizaron las operaciones de cifrado y descifrado correspondientes.

El código en Python para ejecutar este ejemplo es el siguiente:

```
print(f"Clave pblica: {clave_publica}")
print(f"Clave privada: {clave_privada}")
print(f"Primos p: {p}, q: {q}")

mensaje = str(input("Ingresa tu palabra: "))
print(f"Mensaje original: {mensaje}")

cifrado = cifrar(mensaje, clave_publica)
print(f"Mensaje cifrado: {cifrado}")

descifrado = descifrar(cifrado,
    clave_privada, p, q)
print(f"Mensaje descifrado: {descifrado}")
```

Al ejecutarse este código, la consola muestra los siguientes resultados:

```
Clave pública: (65537, 204641772186773503572044570
Clave privada: (170153340097985287038586641853815
Primos p: 24315447591761229086321920159719177644 9,
Ingresa tu palabra: tomate
Mensaje original: tomate
Mensaje cifrado: 18034067292546515649809041209776 5
Mensaje descifrado: tomate
```

Como se puede observar, el sistema cifra correctamente el mensaje original y luego lo descifra, recuperando el mensaje inicial “tomate”. Esto demuestra la funcionalidad y la correcta implementación del algoritmo RSA, incluyendo el uso de optimizaciones como el Teorema del Resto Chino para mejorar el rendimiento en el descifrado.

VIII. Conclusiones

La implementación presentada del algoritmo RSA incorpora varias optimizaciones clave, incluyendo el uso del algoritmo de cuadrado y multiplicación para la exponenciación modular y el Teorema del Resto Chino para el descifrado. Estas técnicas mejoran significativamente la eficiencia del sistema, especialmente para claves de gran tamaño.

Referencias

[1] N. Autor, “Criptografía,” in *Academia.edu*, 2015, accedido: Sep. 23, 2024. [Online]. Available: https://www.academia.edu/download/39531350/15_criptografia.pdf

[2] ——, “Revista de matemáticas superiores,” *Matesup*, 2007, accedido: Sep. 23, 2024. [Online]. Available: http://www.matesup.cl/portal/revista/2007/4.pdf

[1]. [2].