**A Matemática por Trás do Sistema RSA**

**Autores:**

Andrey de Freitas Souza RA: 823217536

Gabrielle Garcia Paz RA: 823126085

Bianca Alves Ribeiro RA: 8222240261

Bruno de Oliveira Santos RA: 823223513

Webster Diógenes Rodrigues RA: 8222242764

**Introdução**

O RSA (*Rivest–Shamir–Adleman*) é um dos algoritmos de criptografia assimétrica mais amplamente utilizados no mundo, sendo empregado em várias aplicações de segurança, incluindo a transmissão de dados de maneira segura. A principal vantagem do RSA é a utilização de um par de chaves: uma pública e uma privada. Este artigo tem como objetivo explicar a matemática envolvida na implementação do algoritmo RSA.

**Sistemas de criptografia**

Os sistemas de criptografia são fundamentais para a segurança digital, e o RSA é uma das escolhas mais populares por suas características únicas. Diferente de sistemas de criptografia simétrica, como o AES, que utiliza uma única chave para criptografar e descriptografar os dados, o RSA é um sistema de criptografia assimétrica. Ele opera com um par de chaves: uma pública, usada para criptografar, e uma privada, usada para descriptografar. Essa abordagem elimina a necessidade de compartilhar uma chave secreta, reduzindo o risco de comprometimento.

O RSA se destaca devido à sua base matemática na fatoração de grandes números primos, o que torna extremamente difícil para um atacante deduzir a chave privada a partir da pública. Além disso, o RSA é amplamente usado em situações como assinaturas digitais e certificados SSL/TLS, que garantem a autenticidade e integridade das comunicações. Embora seja mais lento e menos eficiente para grandes volumes de dados quando comparado ao AES, o RSA é ideal para a troca inicial de chaves em sistemas híbridos, onde o AES pode ser usado para criptografar grandes volumes de dados rapidamente.

**Criptografia RSA**

**1. Geração de Números Primos**

A criptografia RSA depende da multiplicação de dois grandes números primos *p* e *q* para gerar uma chave segura. O método de geração de números primos pode ser descrito matematicamente da seguinte forma:

1. Escolhem-se dois números primos grandes *p* e *q*.
2. Calcula-se o produto *n*, onde:

*n* = *p* × *q*

Esse valor *n* será utilizado tanto na chave pública quanto na chave privada.

No código, esses primos são gerados com a função gerarNumeroPrimo(). A função utiliza a probabilidade para verificar se um número é primo, e se não for, continua a incrementar o valor até encontrar um número que satisfaça a condição de primalidade. Além disso, o código verifica se os números primos gerados são diferentes entre si.

public BigInteger gerarCandidatoPrimo(int tamanho) {

    SecureRandom random = **new** SecureRandom();

    BigInteger p = **new** BigInteger(tamanho, random);

    // Garantir que o número é ímpar e tenha o comprimento correto

    p = p.setBit(tamanho - 1); // Garante que tem o tamanho correto

    p = p.setBit(0);           // Garante que é ímpar

    return p; // Retorna um  número ímpar

}

// Gera um número primo de comprimento especificado em bits

public BigInteger gerarNumeroPrimo(int tamanho) {

    BigInteger p = gerarCandidatoPrimo(tamanho);

    while (!p.isProbablePrime(100)) {  // Verifica se é primo com alta precisão

        p = p.add(BigInteger.TWO);     // Se não for primo, incrementa por 2

    }

    return p; // Retorna o número primo

}

do {

    p = gerarNumeroPrimo(tamanho / 2); // Gera p

    q = gerarNumeroPrimo(tamanho / 2); // Gera q

}while (p.toString().equals(q.toString())); // Continue até que p e q sejam diferentes

**2. A Função Totiente de Euler ϕ(*n*)**

A função totiente de Euler ϕ(*n*) (ou função φ de Euler) é crucial para garantir a segurança do sistema RSA. Ela é definida como o número de inteiros positivos menores que *n* que são coprimos com *n*, ou seja, que possuem máximo divisor comum (MDC) igual a 1.

Para *n* = *p* × *q*, onde *p* e *q* são primos, a função ϕ(*n*) é calculada como:

ϕ(n)=(p−1)×(q−1)

Isso ocorre porque, para cada número primo *p*, ϕ(p)=p−1, já que todos os números menores que um primo são coprimos com ele.

No código, a função totiente é calculada na linha:

BigInteger phi\_n = (p.subtract(BigInteger.ONE)).multiply(q.subtract(BigInteger.ONE));

**3. Geração das Chaves**

Com *n* e ϕ(*n*) calculados, podemos gerar o par de chaves.

* A **chave pública** é composta por dois valores: *e* (um número primo pequeno, geralmente 65537) e *n*:

Chave pública=(*e,n*)

* A **chave privada** é composta por *d*, que é calculado como o **inverso multiplicativo modular** de *e* em relação a ϕ(*n*), ou seja:

*d* × *e* ≡ 1 mod  ϕ(*n*)

Isso significa que *d* é o valor que, quando multiplicado por *e*, resulta em 1 no módulo ϕ(n).

No código, *d* é calculado da seguinte forma:

BigInteger d = e.modInverse(phi\_n);

Aqui, a função modInverse() implementa o **algoritmo de Euclides Estendido** para encontrar *d*, garantindo que a multiplicação entre *d* e *e* seja congruente a 1, ou seja,  
*d* × *e* ≡ 1 mod  ϕ(n).

BigInteger e = BigInteger.valueOf(65537);

BigInteger d = e.modInverse(phi\_n);

**4. Criptografia**

A criptografia no RSA é baseada na exponenciação modular. Quando o remetente deseja enviar uma mensagem para o destinatário, ele criptografa a mensagem M (convertida em número inteiro) usando a chave pública (*e,n*).

A criptografia é dada pela fórmula:

*C* = *M*e mod *n*

Onde *C* é o texto cifrado.

No código, isso é implementado da seguinte forma:

textoPlanoInt.modPow(e, n);

A função modPow() calcula *M*e mod *n* de maneira eficiente, evitando que o número cresça exponencialmente.

**5. Descriptografia**

A descriptografia é feita de maneira semelhante à criptografia, mas usando a chave privada (*d*,*n*). Para recuperar o texto plano *M*, aplicamos:

*M* = Cd  mod  *n*

No código, isso é implementado como:

BigInteger textoPlanoInt = ciphertext.modPow(d, n);

O algoritmo utiliza a propriedade matemática da exponenciação modular e o fato de que *d* × *e* ≡ 1 mod  ϕ(*n*) para garantir que a mensagem original possa ser recuperada.

**6. Exemplo de Execução**

Para ilustrar o processo, considere o seguinte fluxo:

1. **Geração das Chaves**:
   * *p* = 61, *q* = 53
   * *n* = 61 × 53 = 3233
   * ϕ(*n*) = (61 − 1) × (53 – 1) = 3120
   * Escolhe-se *e* = 65537
   * Calcula-se *d* como *d*=e−1  mod  3120 = 2753
2. **Criptografia**:
   * Mensagem *M* = 65
   * Texto cifrado *C* = 6565537 mod  3233 = 2790
3. **Descriptografia**:
   * Texto cifrado *C* = 2790
   * Mensagem recuperada *M* = 27902753 mod 3233 = 65

**Conclusão**

O RSA é um sistema criptográfico robusto, baseado em problemas matemáticos difíceis, como a fatoração de grandes números primos. A segurança do RSA depende do fato de que é fácil multiplicar grandes números primos, mas extremamente difícil fatorar o produto deles. Entender a matemática por trás do RSA, como a função totiente de Euler e o cálculo do inverso modular, é fundamental para compreender como ele protege a comunicação digital.

**Referências**

<https://www.techrepublic.com/article/asymmetric-vs-symmetric-encryption>

<https://www.comparitech.com/blog/information-security/rsa-encryption/>

<https://brilliant.org/wiki/rsa-encryption/>