UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
COORDENAÇÃO DO CURSO DE LICENCIATURA EM MATEMÁTICA

JOSE AUGUSTO DE SOUZA

CRIPTOGRAFIA RSA

TRABALHO DE CONCLUSÃO DE CURSO

TOLEDO
2018

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
COORDENAÇÃO DO CURSO DE LICENCIATURA EM MATEMÁTICA

JOSE AUGUSTO DE SOUZA

CRIPTOGRAFIA RSA

Trabalho de Conclusão de Curso apresentado ao curso de Licenciatura em Matemática da Universidade Tecnológica Federal do Paraná, Câmpus Toledo, como requisito parcial à obtenção do título de Licenciado em Matemática.

Orientador: Adriano Gomes de Santana

TOLEDO
2018

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
COORDENAÇÃO DO CURSO DE LICENCIATURA EM MATEMÁTICA

TERMO DE APROVAÇÃO

O Trabalho de Conclusão de Curso intitulado "Criptografia RSA" foi considerado **APROVADO** de acordo com a ata nº __ de __/__/____

Fizeram parte da banca examinadora os professores:

Prof. Me. Adriano Gomes de Santana

Profª. Me. Larissa Hagedorn Vieira

Prof. Dr. Robson Willians Vinciguerra

Prof. Dr. Wilian Francisco de Araujo

TOLEDO
2018

RESUMO

Neste trabalho, estudamos a criptografia RSA, bem como os conteúdos matemáticos da teoria dos números envolvidos nesse método de criptografia. Com exemplos, utilizamos o método para mostrar como codificar e decodificar uma mensagem, estudamos os algoritmos envolvidos e alguns algoritmos de testes de primalidade e fatoração, assim como o seus custos, buscando justificar a segurança da criptografia RSA.

Palavras-chave: Criptografia RSA. Aritmética Modular. Custo Assintótico. Algoritmos de Fatoração. Testes de Primalidade.

ABSTRACT

In this paper, we study the RSA cryptography as well as the mathematical contents of the number theory involved. With examples, we use the method to show how to encode and decode a message. We also studied the algorithms involved and some algorithms for primality and factorization tests, as well as their costs, seeking to justify the security of RSA encryption.

Keywords: RSA Encryption. Modular Arithmetic. Asymptotic Cost. Factoring Algorithms. Primality Tests.

SUMÁRIO

1 INTRODUÇÃO

A palavra criptografia se origina das palavras gregas *kryptós* que significa "secreto" ou "oculto", e *graphein* que significa "escrita". Deste modo, podemos definir criptografia como a ciência que estuda métodos para transformar uma mensagem em códigos secretos (COUTINHO, 2015, pg. 1). Um método de criptografia possui dois algoritmos, o algoritmo de codificação que é o método pelo qual um texto é transformado em código e o algoritmo de decodificação cuja função é obter o texto original a partir do código gerado pelo algoritmo anterior. O mais simples desses métodos consiste em trocar cada letra da mensagem por uma outra letra ou símbolo. O imperador romano Julio César utilizou um método semelhante para comunicar-se com as legiões em combate na Europa. O método utilizado por César é o primeiro exemplo de um código secreto que se tem registro. Este método consiste em trocar cada letra da mensagem pela terceira letra seguinte do alfabeto. Por exemplo, a mensagem "JOSE" codificada de acordo com o método de César seria "MRVH", já o algoritmo de decodificação seria trocar cada letra pela terceira letra anterior do alfabeto.

O algoritmo de codificação utilizado por César pode ser entendido como uma função bijetora do alfabeto nele mesmo. Substituindo o alfabeto por números de 1 a 26, na ordem do alfabeto ($a = 1, b = 2, c = 3, \ldots, z = 26$), podemos dizer que a função utilizada por César em seu algoritmo foi

$$f : \{1, 2, 3, \ldots, 26\} \to \{1, 2, 3, \ldots, 26\}$$
$$f(x) = \begin{cases} x + 3 & \text{se} \quad x \in \{1, 2, 3, \ldots, 23\} \\ x - 23 & \text{se} \quad x \in \{24, 25, 26\} \end{cases}$$

Este algoritmo também pode ser representado da seguinte forma:

Algoritmo 1.1 *Algoritmo de César*

Entrada: *Texto Original.*

Saída: *Texto Criptografado.*

Passo 1: *Considere a Primeira Letra do Texto;*

Passo 2: *Tome n a posição da letra considerada;*

Passo 3: *Determine o resto r da divisão de $n + 3$ por 26;*

Passo 4: *Coloque no texto codificado a letra que ocupa a posição r no alfabeto;*

Passo 5: *Se não há mais letras no texto original, pare;*

Passo 6: *Considere a próxima letra do texto original e volte para o* **Passo 2**.

Para decodificar a mensagem, basta utilizar a inversa de f. A história registra ainda outros métodos de criptografia semelhantes, baseados em troca de letras por outras letras ou símbolos, matrizes, números primos, curvas matemáticas entre outros.

"A criptografia tem uma irmã gêmea na arte de decifrar códigos secretos, a criptoanálise. Naturalmente todo código vem acompanhado de duas receitas: uma para codificar uma mensagem; outra para decodificar uma mensagem codificada."(COUTINHO, 2009, pg. 1) É importante salientar que decodificar é diferente de decifrar. Quando codificamos algo, apenas o destinatário legítimo consegue decodificar utilizando a chave de decodificação. Entretanto, se alguém intercepta a mensagem e tenta quebrar o código, este alguém está decifrando a mensagem, mesmo desconhecendo a chave de decodificação inicialmente. Ou seja, a criptoanálise estuda os pontos fracos dos métodos de criptografia.

Um método de criptografia pode ser classificado como método de chave privada ou pública. São exemplos de criptografias de chave pública a criptografia de curvas elípticas e a criptografia RSA (HOFFSTEIN et al., 2008). No caso da criptografia RSA, o problema matemático envolvido que torna o método eficiente é a dificuldade de fatoração.

No segundo capítulo deste trabalho está contido os resultados provenientes da aritmética modular, resultados indispensáveis para o estudo do método RSA, assim como sua eficiência e segurança. No terceiro capítulo, apresentamos os parâmetros, algoritmos de codificação e decodificação do RSA, assim como um exemplo prático de codificação e decodificação. No quarto capítulo definimos o custo assintótico de algoritmos, e também os algoritmos das quatro operações básicas, assim como o algoritmo da potenciação em $\mathbb{Z}_n$ e seus respectivos custos. O quinto capítulo trata de alguns métodos de fatoração de um número, e também testes de primalidade, além de seus respectivos custos assintóticos. No último capítulo deste trabalho falamos sobre a segurança do método, ressaltando conceitos contidos nos capítulos anteriores que garantem a segurança do método RSA.

2 ARITMÉTICA MODULAR

Para implementar o algoritmo de criptografia RSA é necessário trabalhar com operações que envolvem restos de divisões, que na maioria das vezes são números "relativamente grandes". Tal trabalho só é possível com os conceitos de aritmética modular. Antes de definir congruência, é necessário conhecer o Algoritmo da Divisão de Euclides.

Teorema 2.1 (Algoritmo da Divisão de Euclides) *Sejam a e b inteiros positivos. Existem números inteiros q e r tais que*

$$a = bq + r \quad 0 \leq r < b$$

Além disso, os valores de q e r satisfazendo as relações acima são únicos.

Prova: Considere o seguinte conjunto:

$$S = \{a - bx; x \in \mathbb{Z}, a - bx \geq 0\}$$

Afirmação 1: $S \neq \emptyset$;

De fato, tome $x = 0$, assim $a - b \cdot 0 = a \geq 0$ por definição. Portanto $a \in S$.

Afirmação 2: S possui elemento mínimo;

Com efeito, como $a - bx \geq 0$, temos que S é limitado inferiormente. Pelo princípio da boa ordem (todo subconjunto de $\mathbb{Z}$ limitado inferiormente possui elemento mínimo), S possui mínimo. Seja r o mínimo de S. Como $r \in S$, então existe $q \in \mathbb{Z}$ tal que $r = a - bq$.

Afirmação 3: $0 \leq r < b$.

De fato. Se $r \geq b$, então $a - bq \geq b$, ou seja $a - bq - b \geq 0$, isto é $a - b(q+1) \geq 0$. Logo $a - b(q+1) \in S$. Observe que $a - b(q+1) = a - bq - b = r - b < r$. Absurdo! Pois r é o elemento mínimo do conjunto S. Logo $0 \leq r < b$. Assim, existem $q, r \in \mathbb{Z}$ tais que $a = bq + r$ onde $0 \leq r < b$.

Para mostrar a unicidade de q e r, suponhamos que existem $q_1, q_2, r_1, r_2 \in \mathbb{Z}$ tais que $a = bq_1 + r_1$ e $a = bq_2 + r_2$, com $0 \leq r_1, r_2 < b$.

Se $r_1 \leq r_2$, então $0 \leq r_2 - r_1 < b$. Como $a = a$, então

$$\begin{aligned} bq_1 + r_1 &= bq_2 + r_2 \\ bq_1 - bq_2 &= r_2 - r_1 \\ 0 \leq b(q_1 - q_2) &= r_2 - r_1 < b \\ 0 \leq b(q_1 - q_2) &< b \\ 0 \leq q_1 - q_2 &< 1 \end{aligned} \tag{2.1}$$

Como q_1 e q_2 são inteiros, então

$$q_1 = q_2 \tag{2.2}$$

Análogo para $r_1 \geq r_2$.

Substituindo (2.2) em (2.1), temos que $bq_1 - bq_1 = r_2 - r_1$, ou seja $0 = r_2 - r_1$, logo $r_1 = r_2$.

□

Embora tenhamos mostrado a validade do Teorema 2.1 para números inteiros positivos, ele valerá também para valores negativos.

Definição 2.2 *Seja $x \in \mathbb{R}$, definimos $\lfloor x \rfloor$ como maior inteiro que é menor do que x.*

Teorema 2.3 *Se $n, b \in \mathbb{N}$ com $b \geq 2$, então existem e são únicos $n_0, n_1, ..., n_N \in \mathbb{Z}$ com $0 \leq n_i < b$ tais que*

$$n = n_0 + n_1 b + n_2 b^2 + ... + n_{N-1} b^{N-1} + n_N b^N$$

onde $N = \lfloor \log_b n \rfloor$.

Prova: Pelo Teorema 2.1, existem q_0 e n_0 tais que $n = bq_0 + n_0$, com $0 \leq n_0 < b$.

Se $q_0 \neq 0$, então existem q_1 e n_1 tais que $q_0 = bq_1 + n_1$, com $0 \leq n_1 < b$. Dessa forma obtemos a seguinte sequência de equações:

$$n = bq_0 + n_0 \tag{2.3}$$
$$q_0 = bq_1 + n_1 \tag{2.4}$$
$$q_1 = bq_2 + n_2 \tag{2.5}$$
$$\vdots$$
$$q_{N-1} = bq_N + n_N \tag{2.6}$$

Afirmamos que $q_N = 0$ para algum $N \in \mathbb{N}$. De fato, a sequência dos quocientes $q_0, q_1, ..., q_{N-1}$ é decrescente, pois $b \geq 2$. Além disso todos os termos desta sequência são

positivos. Como há um número finito de inteiros positivos menores que q_0, segue que para algum N, teremos $q_{N-1} < b$, assim $q_{N-1} = b \cdot 0 + n_N = n_N$.

Substituindo q_{N-1} da Equação (2.6) na equação anterior, e assim sucessivamente temos que

$$\begin{aligned} n &= b(b(b(...b(n_N) + n_{N-1}) + ...) + n_1) + n_0 \\ &= b_{n_N}^N + b^{N-1}n_{N-1} + ... + b^2 n_2 + bn_1 + n_0 \\ &= n_0 + n_1 b + n_2 b^2 + ... + n_{N-1}b^{N-1} + n_N b^N \end{aligned}$$

A unicidade dos coeficientes $n_0, n_1, ..., n_N$ é dada pela unicidade do quociente e do resto da divisão, que pode ser vista no Teorema 2.1.

Devemos mostrar também que $N = \lfloor \log_b n \rfloor$. Observe que se $N < \lfloor \log_b n \rfloor$, então

$$n = b^N n_N + ... + bn_1 + n_0 \leq b^N(b-1) + ... + b(b-1) + (b-1) \tag{2.7}$$

pois $n_i \leq b - 1$. De (2.7), temos que

$$\begin{aligned} b^N(b-1) + ... + b(b-1) + (b-1) &= (b^N + ... + b + 1)(b-1) \\ &= b^{N+1} - 1 \end{aligned}$$

Observe que $b^{N+1} - 1 < b^{N+1}$. Por sua vez, como estamos supondo que $N < \lfloor \log_b n \rfloor$, então $N + 1 \leq \lfloor \log_b n \rfloor$. Portanto, temos que $n < b^{\lfloor \log_b n \rfloor}$, ou seja, $n < n$. Absurdo!

Caso tivéssemos $N > \lfloor \log_b n \rfloor$, então teríamos

$$n < b^N \leq b^N n_N \leq b^N n_N + \ldots + bn_1 + n_0 = n$$

Portanto $n < n$. Absurdo! Portanto $N = \lfloor \log_b n \rfloor$.

□

Definição 2.4 *Dados os inteiros a e b, dizemos que $m \in \mathbb{Z}$ é o mínimo múltiplo comum de a e b, e denotamos*

$$m = [a, b]$$

se

i. m é um múltiplo comum de a e b;

ii. Se c é um múltiplo de a e b, então $m|c$.

Definição 2.5 *Dados os inteiros a e b, não ambos iguais a zero, dizemos que $d \in \mathbb{Z}$ é o máximo divisor comum de a e b, e denotamos*

$$d = (a, b)$$

se

i. $d > 0$*;*

ii. $d|a$ *e* $d|b$*;*

iii. Se $d'|a$ *e* $d'|b$*, então* $d'|d$*.*

Teorema 2.6 *Se* $a, b \in \mathbb{Z}$*, não ambos iguais a zero, então*

$$a \cdot b = (a, b) \cdot [a, b].$$

A demonstração do Teorema 2.6 pode ser consultada em (HEFEZ, 2011, pg. 63).

O Teorema 2.7 abaixo garante a existência do máximo divisor comum de dois inteiros. Além disso, garante ainda quando existem soluções para equações da forma $ax + by = c$, com $a, b, c \in \mathbb{Z}$. Equações dessa forma são chamadas de equações diofantinas.

Teorema 2.7 (Teorema de Bezout) *Sejam* $a, b, c \in \mathbb{Z}$*, sendo* a *e* b *não ambos nulos, então a equação*

$$ax + by = c \tag{2.8}$$

tem solução se, e somente se $d = (a, b)$ *divide* c*.*

Prova: ($\Rightarrow$) Se (2.8) tem solução x_0, y_0, seja $a = dk_1$ e $b = dk_2$, com $k_1, k_2 \in \mathbb{Z}$, então $c = ax_0 + by_0 = dk_1x_0 + dk_2y_0 = d(k_1x_0 + k_2y_0)$, logo $d|c$.

($\Leftarrow$) Suponha que $d|c$, então existe $k \in \mathbb{Z}$ tal que $c = dk$. Assim se existem $x_0, y_0 \in \mathbb{Z}$ tais que $ax_0 + by_0 = d$, então $ax_0k + by_0k = dk = c$. Mostraremos que tais x_0 e y_0 existem. Vamos considerar o seguinte conjunto

$$L = \{ax + by : x, y \in \mathbb{Z} \text{ e } ax + by > 0\} \tag{2.9}$$

O conjunto $L \neq \emptyset$, pois a e b não são ambos nulos, se $a \neq 0$ então basta tomar $x = a$ e $y = 0$ que teremos $ax + by = a^2 > 0$. Se $a = 0$, então $b \neq 0$, portanto basta tomar $x = 0$ e $y = b$ e teremos $ax + by = b^2 > 0$.

Como L é um subconjunto de inteiros e é limitado inferiormente, então pelo princípio da boa ordem, L possui mínimo. Denotaremos esse mínimo por d, logo existem $x_1, y_1 \in \mathbb{Z}$ tal que $d = ax_1 + by_1$. Queremos mostrar que $d = (a, b)$. Para isso, devemos mostrar que:

i. $d > 0$;

De fato, como $d \in L$, então $d > 0$.

ii. $d|a$ e $d|b$

De fato, pelo Teorema 2.1 temos que existem $q, r \in \mathbb{Z}$ com $0 \leq r < d$ tais que $a = qd + r = (ax_1 + by_1)q + r$, logo $r = a - (ax_1 + by_1)q = a(1 - x_1q) + b(-y_1q)$.

Se $r > 0$, então como $r = a(1 - x_1q) + b(-y_1q)$, então $r \in L$. Temos que $r < d$. Absurdo! Pois d é o mínimo de L. Portanto, $r = 0$, ou seja $d|a$. Analogamente $d|b$.

iii. Se $d'|a$ e $d'|b$, então $d'|d$;

Com efeito, se $d'|a$ e $d'|b$ então existem $k_1, k_2 \in \mathbb{Z}$ tais que $a = d'k_1$ e $b = d'k_2$. Observe que $xa + yb = x(d'k_1) + y(d'k_2) = d'(xk_1 + yk_2)$. Como $(xk_1 + yk_2) \in \mathbb{Z}$, então $d'|(xa + yb)$. Tome $x = x_1$ e $y = y_1$. Assim $d'|(x_1a + y_1b) = d$. Portanto $d = x_1a + y_1b$ é o (a, b).

□

Trocando c por d, o Teorema 2.7 garante a existência do máximo divisor comum.

Definição 2.8 *Sejam $n, a, b \in \mathbb{Z}$ com $n \neq 0$. Dizemos que a e b são **congruentes módulo** n e escrevemos*

$$a \equiv b \bmod n$$

se $a - b$ é múltiplo de n.

Teorema 2.9 *Dados $n, a, b \in \mathbb{Z}$, $a \equiv b \bmod n$ se, e somente se, a e b possuem o mesmo resto na divisão por n.*

Prova: ($\Rightarrow$) Se $a \equiv b \bmod n$, então $a - b = nk$ com $k \in \mathbb{Z}$. Digamos que $b = nq + r$ com $0 \leq r < |n|$, então $a = nk + nq + r = n(k + q) + r$ com $0 \leq r < |n|$. Logo, a e b deixam o mesmo resto r na divisão por n.

($\Leftarrow$) Seja $r \in \mathbb{Z}$ com $0 \leq r < |n|$ tal que $a = nq_1 + r$ e $b = nq_2 + r$, então $a - b = nq_1 + r - nq_2 - r = n(q_1 - q_2)$, logo $a \equiv b \bmod n$.

□

Teorema 2.10 *Sejam a, b, c e d inteiros quaisquer e $n \in \mathbb{N}$ fixo. Então, são válidas as seguintes sentenças:*

i. $a \equiv a \bmod n$;

ii. Se $a \equiv b \bmod n$, *então* $b \equiv a \bmod n$;

iii. Se $a \equiv b \bmod n$ *e* $b \equiv c \bmod n$, *então* $a \equiv c \bmod n$.

Prova:

i. Observe que $a - a = 0 = n \cdot 0$, logo pela definição, $a \equiv a \bmod n$.

ii. Temos que $a \equiv b \bmod n$, isto é $a - b = nk$ com $k \in \mathbb{Z}$. observe que multiplicando $a - b = nk$ por -1, obtemos $b - a = n(-k)$. Como $-k \in \mathbb{Z}$, então $b \equiv a \bmod n$.

iii. Temos que $a \equiv b \bmod n$ e $b \equiv c \bmod n$, isto é,

$$a - b = nk_1 \tag{2.10}$$
$$b - c = nk_2 \tag{2.11}$$

com $k_1, k_2 \in \mathbb{Z}$.

De (2.11) temos que

$$b = nk_2 + c \tag{2.12}$$

Substituindo (2.12) em (2.10), temos que

$$\begin{aligned} a - (nk_2 + c) &= nk_1 \\ a - nk_2 - c &= nk_1 \\ a - c &= nk_1 + nk_2 \\ a - c &= n(k_1 + k_2) \end{aligned}$$

Como $k_1 + k_2 \in \mathbb{Z}$, então $a \equiv c \bmod n$.

□

Teorema 2.11 *Dados $n, a, b, c, d \in \mathbb{Z}$ com $a \equiv b \bmod n$ e $c \equiv d \bmod n$. Então valem as seguintes propriedades:*

i. $a + c \equiv b + d \bmod n$*;*

ii. $ac \equiv bd \bmod n$*;*

iii. Para todo $k \in \mathbb{N}, a^k \equiv b^k \bmod n$.

Prova:

i. Se $a \equiv b \bmod n$ e $c \equiv d \bmod n$, então existem $k_1, k_2 \in \mathbb{Z}$ onde $a - b = nk_1$ e $c - d = nk_2$. Somando as duas igualdades temos $a - b + c - d = nk_1 + nk_2$, então $(a + c) - (b + d) = n(k_1 + k_2)$. Logo $a + c \equiv b + d \bmod n$.

ii. Se $a \equiv b \bmod n$ e $c \equiv d \bmod n$, então existem $k_1, k_2 \in \mathbb{Z}$ onde $a - b = nk_1$ e $c - d = nk_2$. Assim $a = b + nk_1$ e $c = d + nk_2$. Logo

$ac = (b+nk_1)(d+nk_2) = bd+bnk_2+nk_1d+n^2k_1k_2 = bd+n(bk_2+dk_1+nk_1k_2) = bd+nk$

Temos então que $ac - bd = nk$ com $k \in \mathbb{Z}$, logo $ac \equiv bd \bmod n$.

iii. Se $k = 1$, temos que $a^1 \equiv b^1 \bmod n$. Suponha que para algum $m \in \mathbb{N}$, temos que $a^m \equiv b^m \bmod n$. Sabemos que $a \equiv b \bmod n$ e $a^m \equiv b^m \bmod n$, então pelo item anterior

$$\begin{aligned} a^m \cdot a &\equiv b^m \cdot b \bmod n \\ a^{m+1} &\equiv b^{m+1} \bmod n \end{aligned}$$

□

Teorema 2.12 *Seja $a \in \mathbb{Z}$ e $n \in \mathbb{N}$. Se $(a, n) = 1$, então existe $a' \in \mathbb{Z}$ tal que $aa' \equiv 1 \bmod n$.*

Prova: Temos por hipótese que $(a, n) = 1$, então, pelo Teorema 2.7, a equação diofantina $ax + ny = 1$ admite solução x_0, y_0, isto é

$$\begin{aligned} ax_0 + ny_0 &= 1 \\ ax_0 &= 1 - ny_0 \\ ax_0 &= n(y_0) + 1 \\ ax_0 &\equiv 1 \bmod n \end{aligned}$$

portanto, basta tomar $a' = x_0$ e o resultado segue.

□

Um dos resultados mais importantes que utilizaremos na criptografia RSA é o seguinte Teorema.

Teorema 2.13 (Pequeno Teorema de Fermat) *Se p é primo, então para todo $n \in \mathbb{Z}$, $n^p \equiv n \bmod p$.*

Prova: Se $n = 1$, então temos que $1^p = 1 \equiv 1 \bmod p$. Suponha que para algum $k \in \mathbb{N}$, temos que $k^p \equiv k \bmod p$. Temos então que

$$\begin{aligned} (k+1)^p &= \sum_{i=0}^{p} \binom{p!}{i!(p-i)!} k^{p-i} \\ &= k^p + 1 + \sum_{i=1}^{p-1} \binom{p!}{i!(p-i)!} k^{p-i} \\ &= k^p + 1 + pq \end{aligned}$$

Logo $(k+1)^p = k^p + 1 + pq \equiv k^p + 1 \bmod p$, pois pq é múltiplo de p. Observe que $k^p + 1 \equiv k + 1 \bmod p$, pela hipótese de indução. Portanto, segue que $(k+1)^p \equiv k + 1 \bmod p$.

□

Corolário 2.13.1 *Se p é primo e p não divide n, então $n^{p-1} \equiv 1 \bmod p$.*

Prova: Do Teorema 2.13 temos que

$$n^p \equiv n \bmod p. \tag{2.13}$$

Pelo Teorema 2.12, existe n' tal que $nn' \equiv 1 \bmod p$. Multiplicando n' em (2.13), temos que $n^p n' \equiv nn' \bmod p$, observe que isto equivale a $n^{p-1}nn' \equiv nn' \bmod p$, portanto $n^{p-1} \equiv 1 \bmod p$.

□

O Teorema 2.10 nos diz que a definição de congruência módulo n é uma relação de equivalência. Isso nos permite definir classe de equivalência e conjunto quociente por esta relação como segue.

Definição 2.14 *Dado $n \in \mathbb{Z}$ com $n \neq 0$, definimos a* ***classe de congruência*** *de um inteiro a como*

$$\overline{a} = \{b \in \mathbb{Z} : b \equiv a \bmod n\}.$$

Definimos ainda o ***conjunto quociente*** *módulo n como*

$$\mathbb{Z}_n = \{\overline{0}, \overline{1}, \overline{2}, \ldots, \overline{n-1}\}.$$

Dados $\overline{a}$, $\overline{b} \in \mathbb{Z}_n$ definimos a adição e multiplicação de $\overline{a}$ com $\overline{b}$ por $\overline{a} + \overline{b} = \overline{a+b}$ e $\overline{a}\overline{b} = \overline{ab}$. Embora essas definições dependam dos representantes a e b das classes de equivalência, o resultado final não dependerá. A prova disso pode ser vista em (COUTINHO, 2009).

Além disso, pelo Teorema 2.11, mas não só por ele, o conjunto $\mathbb{Z}_n$ com estas operações é uma estrutura algébrica denominada anel. Se ainda $n = p$ é um número primo, pelo Teorema 2.12, $\mathbb{Z}_p$ assume uma estrutura denominada corpo.

Definição 2.15 *Seja $n \in \mathbb{Z}$ com $n \neq 0$. Definimos*

$$\mathbb{U}(n) = \{x \in \mathbb{Z}_n : (x, n) = 1\}.$$

Observe que $\mathbb{U}(n)$ representa o conjunto das classes de equivalência que possuem inverso multiplicativo em $\mathbb{Z}_n$. De fato, se $\overline{a} \in \mathbb{U}(n)$ então $(a, n) = 1$ logo pelo Teorema 2.12, existe $a' \in \mathbb{Z}$ tal que $\overline{a}\overline{a'} = \overline{1}$.

Teorema 2.16 *Seja $a \in \mathbb{U}(n)$. Então existe $k \in \mathbb{Z}$ positivo tal que $(\overline{a})^k = \overline{1}$.*

Prova: Observe que $\mathbb{U}(n)$ é um conjunto finito, pois $\mathbb{U}(n) \subset \mathbb{Z}_n$ e pela Definição 2.14, $\mathbb{Z}_n = \{\overline{0}, \overline{1}, \overline{2}, \ldots, \overline{n-1}\}$. Assim, a sequência $\overline{a}, (\overline{a})^2, \ldots, (\overline{a})^n, \ldots$ possuem termos repetidos em $\mathbb{U}(n)$. Seja $i, j \in \mathbb{N}^*$ tais que $i \neq j$ e $(\overline{a})^i = (\overline{a})^j$. Suponha, sem perda de generalidade que $i < j$ e seja $k = j - i$, então $k > 0$, além disso

$$(\overline{a})^k = (\overline{a})^{j-i} = (\overline{a})^j \cdot (\overline{a'})^i = (\overline{a})^j((\overline{a})^i)' = (\overline{a})^j((\overline{a})^j)' = \overline{1}$$

□

Definição 2.17 *Seja $a \in \mathbb{U}(n)$. Dizemos que $k \in \mathbb{Z}$ é a ordem de $\overline{a}$ em $\mathbb{U}(n)$, se $(\overline{a})^k = \overline{1}$ e k é o menor inteiro positivo que satisfaz isto.*

Teorema 2.18 (Teorema de Lagrange) *Seja a um elemento de $\mathbb{U}(n)$ com ordem j, onde n é um natural maior que 1. Se $a^k = \overline{1}$, então j divide k.*

Prova: Pelo Teorema 2.1, existem inteiros q e r tais que $k = qj + r$, com $0 \leq r < j$. Portanto

$$\overline{1} = a^k = a^{qj+r} = (a^j)^q a^r = (\overline{1})^q a^r = a^r.$$

Pela Definição 2.17 concluímos que $r = 0$ e, portanto, j divide k.

□

Definição 2.19 *A função ϕ de Euler de um natural n é definida como:*

$$\phi(n) = |\{k : 1 \leq k \leq n \text{ e } (k, n) = 1\}| = |\mathbb{U}(n)|.$$

Teorema 2.20 *Se p é primo e $k \in \mathbb{N}$, então*

$$\phi(p^k) = p^{k-1}(p-1).$$

Prova: Seja $a \in \mathbb{Z}_{p^k}$. Se $(a, p) \neq 1$, então $p|a$, pois p é primo. Queremos saber quantos elementos $a \in \mathbb{Z}_{p^k}$ existem tais que $(a, p) \neq 1$, ou seja, tais que $p|a$. Tais múltiplos de p são

$$1p, 2p, 3p, \ldots, p^{k-2}p, \ldots, p^{k-1}p = p^k = \overline{0}$$

ou seja, existem p^{k-1} elementos $a \in \mathbb{Z}_{p^k}$ tais que $(a, p) \neq 1$, logo existem $p^k - p^{k-1}$ elementos $a \in \mathbb{Z}_{p^k}$ tais que $(a, p) = 1$. Portanto,

$$\phi(p^k) = p^k - p^{k-1} = p^{k-1}(p-1).$$

□

Vamos calcular como exemplo $\phi(81)$. Note que $\mathbb{U}(81) = \mathbb{U}(3^4)$, ou seja, $p = 3$ e $k = 4$. Primeiro vamos encontrar os elementos $a \in \mathbb{U}(81)$ tais que $(a, 3) \neq 1$, ou seja, os múltiplos de 3. Os múltiplos de 3 em $\mathbb{U}(81)$ são

$$\underbrace{\overline{1 \cdot 3}, \overline{2 \cdot 3}, \overline{3 \cdot 3}, \overline{4 \cdot 3}, \ldots, \overline{25 \cdot 3}, \overline{26 \cdot 3}, \overline{27 \cdot 3} = \overline{0}}_{27 \text{ elementos}}$$

Portanto, $\mathbb{U}(81)$ possui 27 múltiplos de 3. Como $\phi(81)$ é o número de elementos $a \in \mathbb{U}(81)$ tais que $(a, 3) = 1$, então $\phi(81) = 81 - 27 = 54$. Vamos observar essa subtração que representa $\phi(81)$ de outra maneira

$$\begin{aligned} \phi(81) &= 81 - 27 \\ &= 3^4 - 3^3 \\ &= 3^3(3 - 1) \\ &= 3^{4-1}(3 - 1) \end{aligned}$$

que é exatamente o que o Teorema 2.20 nos diz.

Teorema 2.21 *Se m, n são inteiros positivos tais que $(m, n) = 1$, então*

$$\phi(mn) = \phi(m)\phi(n).$$

Prova: Sejam $x, y \in \mathbb{Z}$ tais que $0 \leq x < n$ e $0 \leq y < m$, então vamos provar que existe um único a tal que

$$a \equiv x \bmod n \tag{2.14}$$

$$a \equiv y \bmod m \tag{2.15}$$

com $0 \leq a < mn$. De fato. De (2.14) temos que $a - x = nq$ com $q \in \mathbb{Z}$, então

$$a = nq + x. \tag{2.16}$$

Substituindo (2.16) em (2.15) temos que

$$\begin{aligned} nq + x &\equiv y \bmod m \\ nq &\equiv (y - x) \bmod m. \end{aligned}$$

Multiplicando n', tal que $nn' \equiv 1 \bmod m$ temos

$$\begin{aligned} n'nq &\equiv n'(y-x) \bmod m \\ q &\equiv n'(y-x) \bmod m \\ q &= mt + n'(y-x) \end{aligned} \tag{2.17}$$

com $t \in \mathbb{Z}$. Substituindo (2.17) em (2.16) temos

$$\begin{aligned} a &= n(mt + n'(y-x)) + x \\ &= nmt + nn'(y-x) + x \end{aligned} \tag{2.18}$$

Como controlar t para que isto ocorra? Queremos tomar t tal que $0 \leq a < mn$. Da igualdade (2.18) segue que $0 \leq nmt + nn'(y-x) + x < mn$, ou seja

$$\frac{-nn'(y-x)-x}{nm} \leq t < \frac{nm - nn'(y-x) - x}{nm}$$
$$\frac{-nn'(y-x)-x}{nm} \leq t < \frac{nm}{nm} + \frac{-nn'(y-x)-x}{nm}$$

ou seja, considerando $k = \frac{-nn'(y-x)-x}{nm}$ temos que $k \leq t < 1 + k$, que garante a existência e a unicidade de $t \in \mathbb{Z}$.

Observe que $(a, mn) = (nn'(y-x) + x, nm) = 1$ se, e somente se, $(nn'(y-x)+x, n) = (nn'(y-x)+x, m) = 1$. Note que, de (2.14) temos que $nn'(y-x)+x \equiv x \bmod n$. Por outro lado, observe também que $nn'(y-x) + x \equiv (y-x) + x \equiv y \bmod n$ de modo que $(a, mn) = 1$ se, e somente se $(x, n) = (y, m) = 1$.

Sabemos que para cada $0 \leq x < n$ e $0 \leq y < m$ existe um único $0 \leq a < mn$. Como existem $\phi(m)\phi(n)$ valores para x e y respectivamente tais que $0 \leq x < n$ com $(x, n) = 1$ e $0 \leq y < m$ com $(y, m) = 1$, logo existem $\phi(m)\phi(n)$ valores para a, ou seja

$$\phi(mn) = \phi(m)\phi(n)$$

□

Na demonstração do Teorema 2.21, mostramos um caso particular do Teorema Chinês do Resto, que pode ser consultado em (COUTINHO, 2009, pg. 125).

Teorema 2.22 *Dado n um inteiro positivo diferente de 1, e seja $n = p_1^{r_1} p_2^{r_2} \dots p_k^{r_k}$ a fatoração de n em números primos, então*

$$\phi(n) = p_1^{r_1-1} p_2^{r_2-1} \dots p_k^{r_k-1} (p_1 - 1)(p_2 - 1) \dots (p_k - 1).$$

Prova: De fato, como $n = p_1^{r_1} p_2^{r_2} \dots p_k^{r_k}$, então

$$\phi(n) = \phi(p_1^{r_1} p_2^{r_2} \dots p_k^{r_k}). \tag{2.19}$$

Pelo Teorema 2.21, segue que (2.19) equivale a

$$\phi(n) = \phi(p_1^{r_1})\phi(p_2^{r_2}) \dots \phi(p_k^{r_k}). \tag{2.20}$$

Pelo Teorema 2.20, segue que (2.20) equivale a

$$\begin{aligned} \phi(n) &= p_1^{r_1-1}(p_1-1)p_2^{r_2-1}(p_2-1)\dots p_k^{r_k-1}(p_k-1) \\ &= p_1^{r_1-1}p_2^{r_2-1}\dots(p_1-1)(p_2-1)\dots(p_k-1). \end{aligned}$$

□

Teorema 2.23 *Se $a \in \mathbb{U}(n)$, então*

$$a^{\phi(n)} = \overline{1}.$$

Prova: Temos que $a \in \mathbb{U}(n)$. Pelo Teorema 2.18, a ordem k de $\overline{a}$ divide $\phi(n)$, isto é, $\phi(n) = kj$ com $j \in \mathbb{Z}$. Portanto $a^{\phi(n)} = (a^k)^j = (\overline{1})^j = \overline{1}$.

□

Teorema 2.24 *Se p é um primo ímpar e k um natural, então $\mathbb{U}(p^k)$ possui um gerador, isto é, existe $g \in \mathbb{U}(p^k)$ tal que para todo $a \in \mathbb{U}(p^k)$ existe $i \in \mathbb{N}$ tal que $a = g^i$.*

Teorema 2.25 *Seja p um número primo. Se $d|p-1$, então $\mathbb{U}(p)$ possui exatamente $\phi(d)$ elementos de ordem d. Em particular, $\mathbb{U}(p)$ possui $\phi(p-1) \neq 0$ geradores.*

Os Teoremas 2.24 e 2.25 não serão demonstrados neste trabalho, pois fogem um pouco do nosso foco, porém precisamos desses resultados para provar o Teste de Primalidade de Miller. As provas desses teoremas podem ser consultadas em (LEMOS, 2010, pg. 55).

3 ALGORITMO RSA

Digamos que Alice quer enviar uma mensagem m para Bob, para isso eles precisam estabelecer um canal de comunicação, que pode ser um encontro, uma carta enviada pelo correio ou algum mensageiro, ou uma mensagem via internet. Entretanto, os canais que estão disponíveis para Alice e Bob nem sempre são seguros, digamos por exemplo que Eve tenha acesso ao canal de comunicação entre Alice e Bob, no momento em que Alice tentar enviar a mensagem, Eve conseguirá interceptá-la.

Uma solução para esta situação, é Alice codificar m. Porém, ela e Bob precisam combinar como codificarão. O problema é que qualquer conversa entre Alice e Bob, Eve pode ter total acesso pelo canal de comunicação. Veremos que a solução mais adequada é Alice utilizar um algoritmo de chave pública. Veremos mais adiante o que caracteriza um algoritmo de chave pública.

Definição 3.1 *Os elementos de um algoritmo de criptografia são um conjunto M de mensagens, um conjunto C de cifras, os conjuntos K_c e K_d das chaves, a codificação $e : M \times K_c \to C$, a decodificação $d : C \times K_d \to M$ e a propriedade de que para qualquer $k_c \in K_c$ existe $k_d \in K_d$ tal que se $m \in M$, $d(e(m, k_c), k_d) = m$.*

Tabela 3.1: Algoritmo de Criptografia de Chave Pública

Parâmetros Públicos	
$M,\ C,\ K_c,\ K_d,\ e : M \times K_c \to C$ $d : C \times K_d \to M,\ k_c,\ c$	
Alice	**Bob**
Criação das chaves	
	Escolhe um par de chaves $(k_c, k_d) \in (K_c, K_d)$ Envia k_c para Alice
Codificação	
Escreve m Calcula $c = e(m, k_c)$ Envia c para Bob	
Decodificação	
	Calcula $m = d(c, k_d)$

Observe que Eve está a par dos parâmetros públicos, porém, para decodificar a mensagem, é necessário calcular $d(c, k_d)$, mas Eve desconhece a chave de decodificação k_d.

Existem alguns cuidados a serem tomados quando vamos criptografar algo para que a criptografia seja segura:

- deve ser relativamente fácil calcular $c = e(m, k_c)$;
- deve ser relativamente fácil calcular $m = d(c, k_d)$;
- dados $c_1, c_2, ..., c_n \in C$ não deve ser fácil encontrar $d(c_1, k_d), d(c_2, k_d), ..., d(c_n, k_d)$ sem a chave k_d;
- dados os pares $(m_1, c_1), (m_2, c_2), ..., (m_n, c_n)$ usando a chave k_c deve ser inviável decodificar um texto c.

Este último item é fundamental para determinar se a criptografia é de chave pública. O fato de ser inviável encontrar k_d dado k_c torna a criptografia um algoritmo de chave pública. Se for possível obter k_d dado k_c, então o método é de chave privada (SANTANA, 2013, pg. 19).

3.1 O ALGORITMO RSA

O algoritmo RSA foi inventado em 1978 por R. L. Rivest, A. Shamir e L. Adleman, que na época trabalhavam no Massachussets Institute of Technology (MIT).

Figura 3.1: R. L. Rivest, A. Shamir e L. Adleman em 2003
Fonte: https://www.usc.edu

Digamos que queremos escrever em forma de código uma mensagem m utilizando o RSA, para isso precisamos:

1. transformar m em um número (pré-codificação);
2. escolher dois números primos p e q (parâmetros);
3. calcular $n = p \cdot q$;
4. escolher k_c tal que $(k_c, (p-1)(q-1)) = 1$;
5. encontrar k_d tal que $k_c k_d \equiv 1 \bmod (p-1)(q-1)$.

De acordo com a Definição 3.1, dizemos que m em forma de um número é a mensagem a codificar, n e k_c é a chave de codificação e n e k_d é a chave de decodificação. Devemos nos atentar ao fato de que o número que representa a mensagem a codificar nunca deve ser maior do que n, caso isso aconteça, devemos separar esse número em blocos b_i, com $i = 1, 2, \ldots, k$ tais que $b_i < n$, e codificar cada bloco separadamente. Veja a seguir os algoritmos de codificação e de decodificação do método RSA.

Algoritmo 3.2 *Algoritmo de codificação RSA*

Entrada: *Blocos b_i e os parâmetros k_c e n.*
Saída: *Blocos r_i de M codificada.*

Passo 1: *Coloque $i \Leftarrow 1$;*
Passo 2: *Calcule r_i tal que $(b_i)^{k_c} \equiv r_i \bmod n$;*
Passo 3: *Se $i = k$ pare;*
Passo 4: *Se $i \neq k$, coloque $i \Leftarrow i + 1$ e volte para o* **Passo 2**.

Algoritmo 3.3 *Algoritmo de decodificação RSA*

Entrada: *Blocos r_i e os parâmetros k_d e n.*
Saída: *Blocos b_i de M decodificada.*

Passo 1: *Coloque $i \Leftarrow 1$;*
Passo 2: *Calcule b_i tal que $(r_i)^{k_d} \equiv b_i \bmod n$;*
Passo 4: *Se $i = k$ pare;*
Passo 4: *Se $i \neq k$, coloque $i \Leftarrow i + 1$ e volte para o* **Passo 2**.

O Teorema 3.4 a seguir garante a propriedade descrita na Definição 3.1 que, nos parâmetros do RSA, para todo $k_c \in K_c$, existe $k_d \in K_d$ tal que se $m \in M$, $d(e(m, k_c), k_d) = m$, garantindo assim o funcionamento do método RSA.

Teorema 3.4 *Nos parâmetros do algoritmo RSA temos que $m^{k_c k_d} \equiv m \bmod n$ para todo m.*

Prova: Como $k_c k_d \equiv 1 \bmod (p-1)(q-1)$, então temos que $k_c k_d = (p-1)(q-1)k + 1$ para algum $k \in \mathbb{Z}$. Então temos que

$$\begin{aligned} m^{k_c k_d} &= m^{(p-1)(q-1)k+1} \\ m^{k_c k_d} &= m^{(p-1)(q-1)k} \cdot m \\ m^{k_c k_d} &= (m^{(q-1)k})^{(p-1)} \cdot m \\ m^{k_c k_d} &= (m^{(p-1)k})^{(q-1)} \cdot m \end{aligned}$$

Seja $a = m^{(q-1)k}$ e $b = m^{(p-1)k}$. Se p divide m então $m^{k_c k_d} \equiv 0 \equiv m \bmod p$.

Analogamente se q divide m, então $m^{k_c k_d} \equiv m \bmod q$. Se p não divide m então não divide $a = m^{(q-1)k}$, portanto $m^{k_c k_d} = a^{p-1} m \bmod p$. Pelo Corolário 2.13.1, então $a^{p-1} m \equiv 1 \cdot m \bmod p$.

De modo análogo provamos que $m^{k_c k_d} = b^{q-1} m \equiv 1 \cdot m \bmod q$.

Ou seja, p e q dividem $m^{k_c k_d} - m$. Sendo p e q primos, ambos são fatores de $m^{k_c k_d} - m$. Ou seja, pq divide $m^{k_c k_d} - m$. Sendo $pq = n$ temos que n divide $m^{k_c k_d} - m$, isto é $m^{k_c k_d} \equiv m \bmod n$.

□

3.2 EXEMPLO DE IMPLEMENTAÇÃO DO RSA

Como exemplo, vamos codificar e decodificar a mensagem $m = JOSE$. Para codificar uma mensagem utilizando o método RSA devemos:

1. transformar m em um número (pré-codificação);

Podemos utilizar a Tabela 3.2 de conversão, colocando o número 99 no lugar dos espaços entre cada palavra para pré-codificar a mensagem $JOSE$. Lembrando que outras tabelas de conversão podem ser utilizadas.

Tabela 3.2: Alfabeto de Pré-Codificação

A	B	C	D	E	F	G	H	I	J	K	L	M
10	11	12	13	14	15	16	17	18	19	20	21	22
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
23	24	25	26	27	28	29	30	31	32	33	34	35

Portanto, a palavra $JOSE$ é convertida no número $M = 19242814$ de acordo com a Tabela 3.2.

2. escolher dois números primos p e q (parâmetros);

Como parâmetros, iremos escolher os números primos $p = 47$ e $q = 97$.

3. calcular $n = p \cdot q$;

No caso, $n = 47 \cdot 97 = 4559$.

4. escolher k_c tal que $(k_c, (p-1)(q-1)) = 1$;

Devemos escolher k_c tal que $(k_c, (47-1)(97-1)) = 1$, ou seja $(k_c, 4416) = 1$. Note que podemos escolher $k_c = 5$, pois $(5, 4416) = 1$.

5. encontrar k_d tal que $k_c k_d \equiv 1 \bmod (p-1)(q-1)$.

Devemos então calcular k_d tal que $5k_d \equiv 1 \bmod 4416$, isto é $5k_d = 4416k + 1$, com $k \in \mathbb{Z}$. Portanto, devemos calcular uma solução para a equação diofantina $5k_d - 4416k = 1$. Pelo Teorema 2.7, existe solução inteira para essa equação se $(4416, 5) = 1$, o que é satisfeito. Assim, podemos encontrar uma solução para esta equação diofantina utilizando o Algoritmo Estendido de Euclides. Este método é apresentado no Algoritmo 4.10.

$$\begin{aligned}(4416, 5) &= (5, 4416 - 883 \cdot 5) \\ &= (5, 1) \\ &= (1, 5 - 4 \cdot 1) \\ &= (1, 1) = 1\end{aligned}$$

Assim, fazendo o caminho inverso dessas operações obtemos:

$$\begin{aligned}1 &= 5 - 4 \cdot 1 \\ &= 5 - 4(4416 - 883 \cdot 5) \\ &= 5 - 4416 \cdot 4 + 3532 \cdot 5 \\ &= 5 \cdot 3533 - 4416 \cdot 4\end{aligned}$$

Portanto, uma solução para essa equação diofantina é $k_d = 3533$ e $k = 4$.

Antes de prosseguir com a codificação da mensagem, devemos observar se $M < n$, no caso $19242814 > 4559$, portanto devemos quebrar M em blocos b_i, tais que $b_i < 4559$. Portanto nossos blocos a codificar são $b_1 = 1924$ e $b_2 = 2814$.

Para codificar o bloco $b_1 = 1924$, devemos determinar r_1 tal que $b_1^{k_c} \equiv r_1 \bmod n$. Assim temos

$$\begin{aligned}1924^5 &\equiv 1924^{2^0+2^2} \bmod 4559 \\ &\equiv (1924^1)(1924^{2^2}) \bmod 4559 \\ &\equiv 1924((1924^2)^2) \bmod 4559 \\ &\equiv 1924(3701776^2) \bmod 4559 \\ &\equiv 1924(4427^2) \bmod 4559 \\ &\equiv 1924 \cdot 19598329 \bmod 4559 \\ &\equiv 1924 \cdot 3747 \bmod 4559 \\ &\equiv 7209228 \bmod 4559 \\ &\equiv 1449 \bmod 4559\end{aligned}$$

Portanto, o bloco b_1 codificado é $r_1 = 1449$.

Analogamente, b_2 codificado é $r_2 = 2329$. Portanto, $JOSE = 19242814$ codificada é 14492329.

Para decodificar o bloco $r_1 = 1449$, devemos calcular b_1 tal que $r_1^{k_d} \equiv b_1 \bmod n$. Observe que isso significa obter o resultado de uma multiplicação de 3533 parcelas módulo n, isto seria inviável caso fizéssemos cada multiplicação por vez, porém podemos reduzir significativamente a quantidade de operações reescrevendo o expoente em sua representação binária. Este método é resumido no Algoritmo 4.9. Assim temos

$$\begin{aligned} 1449^{3533} &\equiv 1449^{2^0+2^2+2^3+2^6+2^7+2^8+2^{10}+2^{11}} \bmod 4559 \\ 1449^{3533} &\equiv 1449^1 \cdot 1449^{2^2} \cdot 1449^{2^3} \cdot 1449^{2^6} \cdot 1449^{2^7} \cdot 1449^{2^8} \cdot 1449^{2^{10}} \cdot 1449^{2^{11}} \bmod 4559 \end{aligned}$$

Para fazer esse cálculo, basta calcular os seguintes resultados de congruência, onde um resultado é obtido a partir do anterior:

$$\begin{aligned} 1449^{2^0} &\equiv 1449 \bmod 4559 & 1449^{2^6} &\equiv 1102 \bmod 4559 \\ 1449^{2^1} &\equiv 2461 \bmod 4559 & 1449^{2^7} &\equiv 1710 \bmod 4559 \\ 1449^{2^2} &\equiv 2169 \bmod 4559 & 1449^{2^8} &\equiv 1781 \bmod 4559 \\ 1449^{2^3} &\equiv 4232 \bmod 4559 & 1449^{2^9} &\equiv 3456 \bmod 4559 \\ 1449^{2^4} &\equiv 2072 \bmod 4559 & 1449^{2^{10}} &\equiv 3915 \bmod 4559 \\ 1449^{2^5} &\equiv 3165 \bmod 4559 & 1449^{2^{11}} &\equiv 4426 \bmod 4559 \end{aligned}$$

Substituindo esses valores, temos que

$$\begin{aligned} 1449^{3533} &\equiv 1449 \cdot 2169 \cdot 4232 \cdot 1102 \cdot 1710 \cdot 1781 \cdot 3915 \cdot 4426 \bmod 4559 \\ &\equiv 3142881 \cdot 4663664 \cdot 3045510 \cdot 17327790 \bmod 4559 \\ &\equiv 1730 \cdot 4366 \cdot 98 \cdot 3590 \bmod 4559 \\ &\equiv 7553180 \cdot 351820 \bmod 4559 \\ &\equiv 3476 \cdot 777 \bmod 4559 \\ &\equiv 2700852 \bmod 4559 \\ &\equiv 1924 \bmod 4559 \end{aligned}$$

Portanto, o bloco $r_1 = 1449$ decodificado é $b_1 = 1924$.

Analogamente, o bloco $r_2 = 2329$ decodificado é $b_2 = 2814$, retornando à mensagem original.

4 ALGORITMOS E CUSTO DE EXECUÇÃO

O custo de um algoritmo depende do tempo necessário para conseguir o resultado. A rapidez do computador é um fator que é levado em conta por uma constante apropriada. O tempo necessário para chegar ao fim de um processo é proporcional ao número de operações a efetuar com os algarismos do número n dado (RIBENBOIM, 2001, pg. 100).

Com base no Teorema 2.3, dizemos que os números são os dígitos da representação de n na base b, e denotamos por $n = (n_N...n_1n_0)_b$. Além disso, n tem exatamente $N = \lfloor \log_b n \rfloor + 1$ dígitos em sua representação na base b.

Antes de formalizarmos o que vem a ser o custo de um algoritmo, vamos mostrar o Algoritmo 4.1 que compara dois inteiros k e l dando como resultado uma das três respostas: $k = l, k < l$ ou $k > l$. Neste algoritmo os inteiros k e l estão representados em uma base b.

Algoritmo 4.1 *Comparação*

Entrada*: dois inteiros* $k = (k_n...k_1k_0)_b$ *e* $l = (l_n...l_1l_0)_b$.

Saída*: ou* $k = l$ *ou* $k < l$ *ou* $l < k$.

Passo 1: *Coloque* $i \Leftarrow n$*;*

Passo 2: *Se* $k_i < l_i$ *pare e escreva "*$k < l$*";*

Passo 3: *Se* $l_i < k_i$ *pare e escreva "*$l < k$*";*

Passo 4: *Se* $i = 0$ *pare e escreva "*$k = l$*", se não coloque* $i \Leftarrow i - 1$ *e volte ao* **Passo 2**.

Note que o Algoritmo 4.1 faz no máximo $n + 1$ comparações com números variando de 0 até $b - 1$.

Vamos supor que exista uma máquina que realize as seguintes operações com inteiros positivos k e l menores que b.

(C) Compara k com l, dizendo quem é maior ou se são iguais;

(A) Soma k com l, dando como resultado o número $(a_1a_0)_b$ onde a_1 é 1 ou 0;

(S) Subtrai k de l caso $l \geq k$;

(M) Multiplica k por l, dando como resultado o número $(a_1a_0)_b$;

(D) Divide um número de 2 dígitos na base b por k, dando como resposta um quociente de 2 dígitos na base b e um resto de 1 dígito.

Suponha também que esta máquina gaste no pior dos casos um tempo igual a $T(C), T(A), T(S), T(M)$ e $T(D)$ para calcular as operações C, A, S, M e D, respectivamente. Assim, note que o custo de execução do Algoritmo 4.1 será, no máximo $(n+1)T(C)$.

Um algoritmo pode envolver mais operações elementares que tornam o cálculo de seu custo um pouco complicado, por isso vamos nos preocupar somente com o comportamento assintótico desse custo dependendo do número de dígitos da entrada. De acordo com (HOFFSTEIN et al., 2008, pg. 76), quando queremos saber se um cálculo é viável, os números $3 \cdot 2^k$ e $100 \cdot 2^k$, por exemplo, significam praticamente a mesma coisa se k for suficientemente grande. Por isso, podemos fazer algumas simplificações quando estamos calculando o custo de um algoritmo, para isso definimos:

Definição 4.2 *Considere as funções $f, g : \mathbb{Z} \to \mathbb{R}$, com $g(n) \geq 0$ $\forall n$. Dizemos que $f(n) = O(g(n))$ se existem constantes n_0 e N tais que*

$$|f(n)| \leq Ng(n)$$

para todo $n \geq n_0$.

Rusticamente falando, a Definição 4.2 nos permite estabelecer um limitante superior para o custo, que de certa forma, simplifica os cálculos. Com o Teorema 4.3, podemos realizar operações de soma e multiplicações utilizando a função O (Definição 4.2).

Teorema 4.3 *Se $f_1 = O(g_1)$ e $f_2 = O(g_2)$, então*

$$\begin{aligned} f_1 + f_2 &= O(\max\{g_1, g_2\}) \\ f_1 \cdot f_2 &= O(g_1 \cdot g_2) \end{aligned}$$

Prova: Por hipótese, existem N_1, N_2, k_1, k_2 tais que

$$\begin{aligned} |f_1(x)| &\leq k_1(g_1(x)) \text{ para todo } x \geq N_1 \\ |f_2(x)| &\leq k_2(g_2(x)) \text{ para todo } x \geq N_2 \end{aligned}$$

Colocando $N = \max\{N_1, N_2\}$ e $k = \max\{k_1, k_2\}$ temos que $|f_1(x)| \leq k(g_1(x))$ e $|f_2(x)| \leq k(g_2(x))$ para todo $x \geq N$, segue que:

$$\begin{aligned} |f_1(x) + f_2(x)| &\leq |f_1(x)| + |f_2(x)| \\ &\leq k(g_1(x)) + k(g_2(x)) \\ &\leq 2k(\max\{g_1(x), g_2(x)\}) \end{aligned}$$

Portanto $f_1 + f_2 = O(\max\{g_1, g_2\})$.

$$\begin{aligned} |f_1(x) \cdot f_2(x)| &\leq |f_1(x)| \cdot |f_2(x)| \\ &\leq k(g_1(x)) \cdot k(g_2(x)) \\ &\leq k^2(g_1(x) \cdot g_2(x)) \end{aligned}$$

Portanto $f_1 \cdot f_2 = O(g_1 \cdot g_2)$.

□

Teorema 4.4 *Dados $n, b \in \mathbb{N}$, com $b > 1$ temos que*

$$O(\lfloor \log_b n \rfloor) = O(\log_a n)$$

para qualquer $a > 1$.

Prova: De fato, observe que pela Definição 4.2, $O(\lfloor \log_b n \rfloor) = O(\log_b n)$ pois $\lfloor \log_b n \rfloor \leq \log_b n$ pela Definição 2.2. Aplicando as propriedades de logaritmo temos que

$$\begin{aligned} O(\log_b n) &= O\left(\frac{\log_a n}{\log_a b}\right) \\ &= O\left(\frac{1}{\log_a b} \cdot \log_a n\right) \\ &= O(\log_a n). \end{aligned}$$

□

Do Teorema 4.4, observe que se $a = e$, temos que $O(\log_b n) = O(\ln n)$. Daqui em diante, sempre que tivermos $O(\lfloor \log_b n \rfloor)$ no cálculo do custo assintótico de um algoritmo, usaremos $O(\ln n)$.

4.1 ALGUNS ALGORITMOS BÁSICOS E SEUS RESPECTIVOS CUSTOS ASSINTÓTICOS

Algoritmo 4.5 *Adição*

Entrada: *dois inteiros* $k = (k_n \dots k_1 k_0)_b$ *e* $l = (l_n \dots l_1 l_0)_b$.

Saída: *o inteiro* $k + l = (s_{n+1} s_n \dots s_1 s_0)_b$.

Passo 1: *Coloque* $r \Leftarrow 0$ *e* $i \Leftarrow 0$;

Passo 2: *Coloque* $(a_1 a_0)_b \Leftarrow k_i + l_1$;

Passo 3: *Se* $r = 0$ *ou* $(r = 1$ *e* $a_0 \neq b - 1)$ *coloque* $s_i \Leftarrow a_0 + r$ *e* $r \Leftarrow a_1$. *Se não coloque* $s_i \Leftarrow 0$ *e* $r \Leftarrow 1$;

Passo 4: *Coloque* $i \Leftarrow i + 1$;

Passo 5: *Se* $i \leq n$ *retorne ao* **Passo 2**, *se não coloque* $s_{n+1} \Leftarrow r$ *e pare.*

Note que o Algoritmo 4.5 tem um custo máximo de $nT(A)$ em cada um dos Passos $2, 3$ e 4, $2nT(C)$ no Passo 2 e $nT(C)$ no Passo 5. Assim, o tempo máximo para a execução este algoritmo é $3nT(A) + 2nT(C) + nT(c) = 3nT(A) + 3nT(C)$, isto é $O(3n(T(A) + T(C))) = O(3n)$ ou seja, $O(n)$.

Como queremos calcular o custo do Algoritmo 4.5 em função do número de dígitos da entrada reescrevemos este custo como $O(\lfloor \log_b k \rfloor)$, ou seja, $O(\ln k)$.

Algoritmo 4.6 *Subtração*

Entrada: *dois inteiros* $k = (k_n \dots k_1 k_0)_b$ *e* $l = (l_n \dots l_1 l_0)_b$ *com* $k \geq l$.

Saída: *o inteiro* $k - l = (s_n \dots s_1 s_0)_b$.

Passo 1: *Coloque* $i \Leftarrow 0$ *e* $h \Leftarrow 0$;

Passo 2: *Se* $h = 0$ *vá para o* **Passo 3**, *se não vá para o* **Passo 4**;

Passo 3: *Se* $k_i \geq l_i$, *coloque* $s_i \Leftarrow k_i - l_i$ *e* $h \Leftarrow 0$. *Se não coloque* $s_i \Leftarrow (b - l_i) + k_i$ *e* $h \Leftarrow 1$;

Passo 4: *Se* $k_i \geq l_i + 1$ *coloque* $s_i \Leftarrow (k_i - l_i) - 1$ *e* $h \Leftarrow 0$. *Se não, coloque* $s_i \Leftarrow ((b - l_i) - 1) + k_i$ *e* $h \Leftarrow 1$;

Passo 5: *Coloque* $i \Leftarrow i + 1$;

Passo 6: *Se* $i \leq n$ *volte ao* **Passo 2**. *Se não, pare.*

Note que o Algoritmo 4.6 tem um custo máximo de $2nT(S) + 2nT(A) + nT(C)$, ou seja, o seu custo assintótico será $O(n)$, ou ainda $O(\lfloor \log_b k \rfloor)$, isto é $O(\ln k)$.

Algoritmo 4.7 *Multiplicação*

Entrada: *dois inteiros positivos* $k = (k_n \dots k_1 k_0)_b$ *e* $l = (l_m \dots l_1 l_0)_b$ *com* $k \geq l$.

Saída: *o inteiro* $kl = (p_{m+n+1} \dots p_1 p_0)_b$.

Passo 1: *Coloque* $p_0 \Leftarrow 0, p_1 \Leftarrow 0, \dots, p+m+n+1 \Leftarrow 0$;

Passo 2: *Coloque* $i \Leftarrow 0, j \Leftarrow 0$ *e* $h \Leftarrow 0$;

Passo 3: *Coloque* $a_1 a_0 \Leftarrow k_j l_i$;

Passo 4: *Coloque* $b_1 b_0 \Leftarrow (a_0 + h) + p_{i+j}$;

Passo 5: *Coloque* $p_{i+j} \Leftarrow b_0$ *e* $h \Leftarrow a_1 + b_1$;

Passo 6: *Se* $i < n$ *coloque* $i \Leftarrow i+1$ *e volte ao* **Passo 3**;

Passo 7: *Se* $j < m$ *coloque* $p_{i+j+1} \Leftarrow h, h \Leftarrow 0, i \Leftarrow 0, j \Leftarrow j+1$ *e volte ao* **Passo 3**. *Se não, pare.*

Observe que, a menos dos índices, o Algoritmo 4.7 leva no máximo um tempo de $mnT(M)$ no Passo 3, $2mnT(S)$ no Passo 4, $mnT(S)$ no Passo 5, portanto o seu custo é de $mnT(M) + 3mnT(S)$, ou seja, $O(mn)$. Se considerarmos $m = n$, temos um custo de $O(n^2) = O(\lfloor \log_b k \rfloor^2)$, ou seja $O(\ln^2 k)$.

No Passo 5 este algoritmo assume que $a_1 + b_1 < b$, tal propriedade deve ser verificada, caso contrário algumas operações nele não poderiam ser feitas.

Note que $(a_1 a_0)_b = k_i l_i \leq (b-1)(b-1) = b(b-2) + 1$, ou seja, $a_1 \leq b-2$. Levando isto em consideração, temos que $b_1 < 2$ quando $a_1 = b-2$.

Se $a_1 = b-2$, então a_0 é no máximo 1, disso segue que $(b_1 b_0)_0 = (a_0 + h) + p_{i+j} \leq (1 + b - 1) + (b-1) = b^2 + (b-1)b^0$, ou seja b_1 só pode ser 1 ou 2.

Se $a_1 \neq b-2$, podemos verificar que $b_1 \leq 2$, e a soma $a_1 + b_1$ sempre será menor que $b-1$.

Algoritmo 4.8 *Divisão*

Entrada: *dois inteiros positivos* $k = (k_m \dots k_1 k_0)_b$ *e* $l = (l_n \dots l_1 l_0)_b$ *com* $k \geq l$.

Saída: *dois inteiros positivos* $q = (q_{m-n} \dots q_1 q_0)_b$ *e* $r = (r_n \dots r_1 r_0)_b$ *tais que* $k = lq + r$ *e* $r < l$.

Passo 1: *Coloque* $q_0 \Leftarrow 0, q_1 \Leftarrow 0, \dots, q_{m-n} \Leftarrow 0$ *e* $i \Leftarrow 0$;

Passo 2: *Coloque* $c \Leftarrow k - lq_i$;

Passo 3: *Encontre* $h \in \{1, 2, \dots, b-1\}$ *tal que* $c_m c_{m-1} \dots c_{m-n-i} \geq lh$ *e* $c_m c_{m-1} \dots c_{m-n-i} < l(h+1)$;

Passo 4: *Coloque* $q_{m-n-i} \Leftarrow h$;

Passo 5: *Coloque* $i \Leftarrow i+1$;

Passo 6: *Se* $i < m-n$ *volte ao* **Passo 2**;

Passo 7: *Coloque* $r \Leftarrow k - lq$ *e pare.*

Apesar de encontrarmos um loop no Passo 3 do Algoritmo 4.8, este não realiza

mais que $b-1$. Além disso, como multiplicamos l por um número que possui apenas 1 dígito na base b, o custo do Passo 3 é de $O(n)$.

Note que este algoritmo utiliza alguns dos algoritmos descritos anteriormente, por exemplo, a multiplicação no Passo 2.

Calculando o custo assintótico por passo, temos:

$$\begin{array}{ll} \text{Passo 2} & (m-n)O(n) + (m-n)O(n(m-n)) \\ \text{Passo 3} & (m-n)O(n) \\ \text{Passo 7} & O(n) + O(n^2) \end{array}$$

Pelo Teorema 4.3, o custo do Algoritmo 4.8 é $(m-n)O(n(m-n))$, ou seja, $O(m^2n + n^3)$. Como estamos supondo que $m \geq n$, então o custo deste algoritmo não ultrapassa $O(m^3) = O(\lfloor \log_b k \rfloor^3)$, isto é, $O(\ln^3 k)$.

Também podemos utilizar estes mesmos algoritmos para fazer adições e multiplicações em $\mathbb{Z}_n$.

Como podemos representar qualquer classe de equivalência de $\mathbb{Z}_n$ por um número entre 0 e $n-1$, o custo da adição e da multiplicação em $\mathbb{Z}_n$ não ultrapassam $O(\ln n)$ e $O(\ln^2 n)$ respectivamente, uma vez que um número entre 0 e $n-1$ não tem mais que $\lfloor \log_b n \rfloor$ dígitos.

Algoritmo 4.9 *Potenciação em* $\mathbb{Z}_n$

Entrada*: dois elementos* $a \in \mathbb{Z}_n$ *e* $k = (k_m, \ldots, k_1 k_0)_2$.

Saída*: o elemento* $a^k \in \mathbb{Z}_n$.

Passo 1: *Coloque* $i \Leftarrow 0, e \Leftarrow 1$ *e* $b \Leftarrow a$*;*

Passo 2: *Se* $k_i = 1$ *coloque* $e \Leftarrow eb$ *em* $\mathbb{Z}_n$*;*

Passo 3: *Coloque* $i \Leftarrow i + 1$*;*

Passo 4: *Coloque* $b \Leftarrow b^2$ *em* $\mathbb{Z}_n$*;*

Passo 5: *Se* $i \geq m$ *volte ao* **Passo 2**. *Se não, o valor de* $a^k = e$.

Note que nos Passos 2 e 4 do Algoritmo 4.9 são feitas m multiplicações em $\mathbb{Z}_n$, além de outras operações com os índices. Portanto, o custo deste algoritmo é $mO(\ln^2 n) = O(\ln k) \cdot O(\ln^2 n)$. Como estamos calculando a^k em $\mathbb{Z}_n$, o valor de k não precisa ser maior do que n. Assim $O(\ln k) \leq O(\ln n)$, de modo que o custo desse algoritmo não ultrapassa $O(\ln n) \cdot O(\ln^2 n) = O(\ln n \cdot \ln^2 n) = O(\ln^3 n)$.

Observe que o resultado encontrado é o resto da divisão do inteiro a^k por n. Embora calcular a^k em $\mathbb{Z}$ e depois dividi-lo por n, com os algoritmos apresentados anteriormente, necessite de uma quantidade exorbitante de cálculos, podemos encontrar o resto da divisão em $\mathbb{Z}_n$ a um custo viável sem encontrar o quociente.

Todos os algoritmos e custos vistos nesta seção até aqui podem ser consultados

com mais informações em (SANTANA; SILVA, 2008).

A seguir, tratamos do Algoritmo Estendido de Euclides, algoritmo fundamental para solucionar uma equação diofantina.

Algoritmo 4.10 *Algoritmo Estendido de Euclides*

Entrada: *dois inteiros* a *e* b *não negativos.*

Saída: *inteiros* u, v, d *tais que* $d = (a, b)$ *e* $au + bv = d$.

Passo 1: *Coloque* $u \Leftarrow 1$, *e* $d \Leftarrow a$;

Passo 2: *Se* $b = 0$ *então* $v \Leftarrow 0$ *e pare;*

Passo 3: *Coloque* $v_1 \Leftarrow 0$ *e* $v_3 \Leftarrow b$;

Passo 4: *Se* $v_3 = 0$, *então* $v \Leftarrow \frac{d-au}{b}$ *e pare;*

Passo 5: *Coloque* $q \Leftarrow \left\lfloor \frac{d}{v_3} \right\rfloor$;

Passo 6: *Coloque* $t_1 \Leftarrow u - qv_1, u \Leftarrow v_1, d \Leftarrow v_3, v_1 \Leftarrow t_1, v_3 \Leftarrow t_3$ *e volte ao* **Passo 4**.

Um algoritmo possui um tempo de execução polinomial se a função que descreve seu custo assintótico é limitada por uma função polinomial do tamanho da entrada $N = \lfloor \log_b n \rfloor$. Um problema matemático é dito polinomial, ou de classe P, se existe um algoritmo que resolve tal problema com custo de execução polinomial. Nos exemplos acima vemos que os problemas de comparação, adição, subtração, multiplicação, divisão, potenciação e o cálculo do inverso multiplicativo em $\mathbb{Z}_n$ são problemas de classe P.

Quando o custo de execução de um algoritmo é polinomial, então é viável a sua implementação de execução em um tempo humano. Um problema é dito não polinomial, ou de classe NP, se qualquer algoritmo que resolve tal problema possui custo de execução maior que qualquer função polinomial da entrada. No próximo capítulo serão apresentados alguns algoritmos cujo tempo de execução é maior que qualquer custo polinomial.

5 FATORAÇÃO E TESTES DE PRIMALIDADE

5.1 FATORAÇÃO

Para trabalhar com números primos, primeiro é necessário encontrá-los, para isso existem métodos matemáticos. O crivo de Eratóstenes é o mais antigo dos métodos para encontrar números primos, e não envolve nenhuma fórmula explícita. Eratóstenes nasceu por volta de 284 a.C. na Grécia. É importante lembrar que crivo significa peneira.

Nicômaco em sua *Aritmética*, publicada por volta de 100 d.C., introduz o crivo de Eratóstenes da seguinte maneira: tomamos os números ímpares misturados de maneira indiscriminada e, por esse método, como se fosse pelo uso de um instrumento ou peneira, separamos os primos ou indecomponíveis dos secundários ou compostos. Portanto, o crivo funciona como uma peneira que só deixa passar os números primos (COUTINHO, 2009, pg. 62).

O crivo determina todos os primos até um certo $n \in \mathbb{N}$ previamente escolhido. Para realizar o crivo, devemos listar os números ímpares até n, em seguida riscamos todos os múltiplos de 3, exceto o próprio 3, em seguida procuramos o menor número da lista diferente de 3 que não tenha sido riscado, no caso o 5, e riscamos todos os múltiplos de 5, exceto o próprio 5, e assim por diante até chegar em $\sqrt{n}$, os números que restarem serão números primos. O Algoritmo 5.1 descreve o método de Eratóstenes.

Algoritmo 5.1 *Crivo de Eratóstenes*

Entrada*: um inteiro positivo n.*

Saída*: a lista de todos os primos $p_1, \ldots, p_r$ menores que n.*

Passo 1: *Coloque $j \Leftarrow 1, r \Leftarrow 1$ e $p_1 \Leftarrow 2$;*

Passo 2: *Coloque $j \Leftarrow j + 2$ e $i \Leftarrow 1$;*

Passo 3: *Se $p_i | j$ volte ao* **Passo 2***;*

Passo 4: *Se $i < r$, coloque $i \Leftarrow i + 1$ e volte ao* **Passo 3***;*

Passo 5: *Se $j < n$ coloque $r \Leftarrow r + 1, p_r \Leftarrow j$ e volte ao* **Passo 2***. Caso contrário, pare.*

A quantidade de números primos p menores que um número dado x pode ser estimada a partir do Teorema 5.2.

Teorema 5.2 (Teorema dos Números Primos) *A quantidade de números primos p, tal que $p \leq x$ é*

$$\pi(x) \sim \frac{x}{\log x}.$$

Um pouco mais sobre o Teorema 5.2 pode ser visto em (RIBENBOIM, 2001, pg. 116) e (HOFFSTEIN et al., 2008, pg. 129).

Determinar se um número é primo não é equivalente ao problema de fatoração. Existem métodos que conseguem verificar se um número é primo ou composto mesmo que não obtenha a fatoração desse número no caso composto. Dois desses métodos são o Teste de Fermat e o Teste de Miller. Este último, embora não verifique que um número é primo com 100% de precisão, a probabilidade de se obter um número primo com esse teste pode ser controlada. Métodos como o Teste de Miller são denominados algoritmos de primalidade probabilísticos. Veremos a seguir um pouco mais sobre esses métodos de fatoração.

5.1.1 FATORAÇÃO DE FERMAT

O Algoritmo 5.3 de fatoração abaixo nos permite encontrar fatores de um número n caso ele seja composto.

Algoritmo 5.3 *Fatoração de Fermat*

Entrada: *um inteiro positivo ímpar* n.

Saída: *um fator de* n *ou uma mensagem indicando que* n *é primo.*

Passo 1: *Comece com* $b = \lfloor \sqrt{n} \rfloor$*; se* $n = b^2$ *então* b *é fator de* n *e podemos parar;*

Passo 2: *Caso contrário incremente* b *de uma unidade e calcule* $a = \sqrt{b^2 - n}$*;*

Passo 3: *Repita o* **Passo 2** *até encontrar um valor inteiro para* a*, ou até que* b *seja igual a* $\frac{n+1}{2}$*: no primeiro caso* n *tem fatores* $a + b$ *e* $a - b$*, no segundo* n *é primo.*

O Algoritmo 5.3 consiste em tentar escrever n como uma diferença de quadrados, isto é, $n = a^2 - b^2 = (a + b)(a - b)$. Se isso acontecer, teremos que $a^2 \equiv b^2 \bmod n$. Além disso, analisando por outro lado, se $a^2 \equiv b^2 \bmod n$, então teremos que $a^2 - b^2 = kn$, com $k \in \mathbb{Z}$. Desta forma

$$\begin{aligned} a^2 - b^2 &= kn \\ (a + b)(a - b) &= kpq \end{aligned}$$

ou seja, se tivermos sorte, $(a + b)$ ou $(a - b)$ pode ser um dos primos p ou q. Vejamos a seguir um exemplo aplicado no Algoritmo 5.3.

Vamos tentar fatorar $n = 273$, ou seja, vamos tentar reescrever 273 como uma diferença de quadrados, isto é, $273 = a^2 - b^2$, ou seja $273 + b^2 = a^2$. No **Passo 1** do algoritmo, calculamos $b = \lfloor \sqrt{273} \rfloor = 16$, porém $16^2 \neq 273$, logo 273 não é um quadrado perfeito. No **Passo 2**, incrementamos b de uma unidade e calculamos $a = \sqrt{b^2 - n}$, isto é, $a = \sqrt{17^2 - 273} = \sqrt{16} = 4$. Portanto, $273 = (17 + 4)(17 - 4) = 21 \cdot 13$. Caso 273 fosse

primo, iriamos repetir o processo até que $b = \frac{273+1}{2} = 137$, e não encontraríamos um valor inteiro para a.

Analisando o Algoritmo 5.3 com o exemplo acima, o que está acontecendo é uma substituição dos valores de b, e verificando se a é um quadrado perfeito, veja

$$273 + 1^2 = 274 \tag{5.1}$$
$$273 + 2^2 = 277 \tag{5.2}$$
$$273 + 3^2 = 282 \tag{5.3}$$
$$273 + 4^2 = 289 \tag{5.4}$$

ou seja, 274, 277 e 282 em (5.1), (5.2) e (5.3) respectivamente, não são quadrados perfeitos, porém em (5.4), $289 = 17^2$ é um quadrado perfeito, por isso $273 + 4^2 = 17^2$, isto é $273 = 17^2 - 4^2 = (17 + 4)(17 - 4)$, encontrando assim, dois fatores de 273.

5.2 TESTES DE PRIMALIDADE

5.2.1 TESTE DE FERMAT

Pelo Corolário 2.13.1 se $n > 0$ e $1 < b < n - 1$ são números inteiros e b^{n-1} não deixa resto 1 quando dividido por n, então n é um número composto. "O número b é conhecido como uma testemunha do fato de n ser composto" (COUTINHO, 2009, pg. 105). Este fato nos permite verificar se um número n é composto.

Um problema desse algoritmo é que nem sempre quando ele concluir que n não é composto, ele realmente não será. Em (COUTINHO, 2009, pg. 106) é dito que Leibniz achava que seria um número primo, ele tomava $b = 2$, que é o caso mais simples de calcular. Mas isso não é verdade, por exemplo $2^{340} \equiv 1 \bmod 341$, portanto 341 seria um número primo, mas $341 = 11 \cdot 31$. Esses "falsos primo" que passam por esse teste são conhecidos como números **pseudoprimos**.

5.2.2 TESTE DE MILLER

Observe que se p for primo, $b \in \mathbb{Z}$ e $p \nmid b$ então, pelo Corolário 2.13.1, $b^{p-1} \equiv 1 \bmod p$. Sejam k e q inteiros positivos com q ímpar tais que $p - 1 = 2^k q$. Desta forma,

$$b^{p-1} - 1 = b^{2^k q} - 1 = ((b^q)^{2^{k-1}} + 1)((b^q)^{2^{k-2}} + 1)...((b^q)^2 + 1)(b^q + 1)(b^q - 1) \equiv 0 \bmod p.$$

Portanto,

$$\begin{aligned} \text{ou } (b^q)^{2^{k-1}} &\equiv -1 \bmod p \\ \text{ou } (b^q)^{2^{k-2}} &\equiv -1 \bmod p \\ &\vdots \\ \text{ou } (b^q)^2 &\equiv -1 \bmod p \\ \text{ou } b^q &\equiv -1 \bmod p \\ \text{ou } b^q &\equiv 1 \bmod p \end{aligned}$$

Substituindo p por um inteiro n, caso uma destas congruências não seja satisfeita, então n será composto. Este fato é base para o Teste de Primalidade de Miller, que está descrito abaixo.

Algoritmo 5.4 *Teste de Miller*

Entrada*: um inteiro ímpar n, e a base b, onde $1 < b < n-1$.*
Saída*: uma das mensagens: 'n é composto' ou 'teste inconclusivo'.*

Passo 1: *Divida $n-1$ sucessivamente por 2 até encontrar q (um número ímpar) e k tais que $n-1 = 2^k q$;*
Passo 2: *Comece fazendo $i = 0$ e $r =$ resto de b^q por n;*
Passo 3: *Se $i = 0$ e $r = 1$ ou se $i \geq 0$ e $r = n-1$ a saída é 'teste inconclusivo';*
Passo 4: *Incremente i de 1 e substitua r pelo resto da divisão de r^2 por n;*
Passo 5: *Se $i < k$ volte ao* **Passo 3***, senão a saída é 'n é composto'.*

A princípio, quando o teste tem saída *inconclusivo*, duas coisas podem acontecer: n pode ser primo, ou n pode ser composto. Vejamos um exemplo a seguir. Vamos aplicar o número 341 ao Teste de Miller. Pelo **Passo 1**, temos que $341 - 1 = 2^2 \cdot 85$. Precisamos calcular as potências de 2 módulo 341 para os expoentes 85 e 170, assim temos que $2^{85} \equiv 32 \bmod 341$ e $2^{170} \equiv 32^2 \equiv 1 \bmod 341$. Portanto, 341 é composto pelo Teste de Miller.

Teorema 5.5 (Teste de Miller) *Seja n um número ímpar composto. Se $n-1 = 2^r m$ para naturais r e m, com m ímpar, então n é pseudoprimo com respeito a no máximo 25% dos elementos pertencentes a $\mathbb{U}(n)$.*

Antes de demonstrar o Teorema 5.5, vamos analisar dois resultados expressos em forma de lema, descritos a seguir.

Lema 5.6 *Seja n um número ímpar livre de quadrados. Se $n-1 = 2^r m$ com m ímpar, $n = p_1 p_2 ... p_k$ a fatoração de n e para cada $i \in \{1, 2, ..., k\}$, $p_i - 1 = 2^{r_i} m_i$ com m_i ímpar. Nestas condições, o número de raízes do polinômio $X^m - \overline{1}$ em $\mathbb{Z}_n$ é $(m, m_1)(m, m_2)...(m, m_k)$, sendo $(a, b) = mdc(a, b)$.*

Prova: Pelo Teorema 2.24, $\exists g_i \in \mathbb{Z}$ tal que $0 \leq g_i < p_i$ e $\overline{g_i}$ gera $\mathbb{U}(p_i)$.

Para todo inteiro a tal que $0 \leq a < n$ e $(a, n) = 1$, existem a_i, com $i = \{1, 2, \ldots, k\}$ e $0 \leq a_i < p_i - 1$ tal que

$$\Psi(\overline{a}) = (\overline{g_1}^{a_1}, \overline{g_2}^{a_2}, ..., \overline{g_k}^{a_k}). \tag{5.5}$$

Agora, caracterizaremos os $a's$ tais que $\overline{a}^m = \overline{1}$ em $\mathbb{Z}_n$. Pela operação de cartesiano de grupos e por (5.5) temos que

$$(\overline{1}, \overline{1}, \ldots, \overline{1}) = \Psi(\overline{1}) = \Psi(\overline{a}^m) = (\overline{g_1}^{a_1 m}, \overline{g_2}^{a_2 m}, \ldots, \overline{g_k}^{a_k m}).$$

Consequentemente esta igualdade será satisfeita se, e somente se $p_i - 1 = 2^{r_i} m_i | a_i m$ para cada $i \in \{1, 2, ..., k\}$. Como $2^{r_i} m_i | a_i m$ então

$$\begin{aligned} a_i m &= 2^{r_i} m_i z \\ a_i &= \frac{2^{r_i} m_i z}{m} \\ &= \frac{2^{r_i} m_i m_i z}{m m_i} \\ &= \frac{2^{r_i} m_i}{(m, m_i)} \cdot \frac{m_i z}{[m, m_i]} \end{aligned}$$

onde $z \in \mathbb{Z}$ e $[m, m_i]$ é o mínimo múltiplo comum entre m e m_i, conforme a Definição 2.4. Portanto, a_i é múltiplo de $\frac{2^{r_i} m_i}{(m, m_i)}$.

Como $0 \leq a_i < p_i - 1$, existem (m, m_i) opções para a_i. Então o número de raízes para $X^m - \overline{1}$ em $\mathbb{Z}_n$ é

$$(m, m_1)(m, m_2)...(m, m_k).$$

□

Lema 5.7 *Seja n um número ímpar livre de quadrados. Se $n-1 = 2^r m$ com m ímpar, $n = p_1 p_2 ... p_k$ a fatoração de n e para cada $i \in \{1, 2, ..., k\}$, $p_i - 1 = 2^{r_i} m_i$ com m_i ímpar. Nestas condições, o número de raízes do polinômio $X^{2^j m} + \overline{1}$ em $\mathbb{Z}_n$ é $2^{kj}(m, m_1)(m, m_2)...(m, m_k)$ quando $j < min\{r_1, ..., r_k\}$ ou 0 caso contrário.*

Prova: Pelo Teorema 2.24, $\exists g_i \in \mathbb{Z}$ tal que $0 \leq g_i < p_i$ e $\overline{g_i}$ gera $\mathbb{U}(p_i)$.

Para todo inteiro a tal que $0 \leq a < n$ e $(a, n) = 1$, existem a_i, com $i = \{1, 2, \ldots, k\}$ e $0 \leq a_i < p_i - 1$ tal que

$$\Psi(\overline{a}) = (\overline{g_1}^{a_1}, \overline{g_2}^{a_2}, ..., \overline{g_k}^{a_k}). \tag{5.6}$$

Para $j \in \{0, 1, ..., r-1\}$, necessitamos descrever os $a's$ tais que $\overline{a}^{2^j m} = \overline{-1}$ em

$\mathbb{Z}_n$. Pela operação de cartesianos de grupos e por (5.6) temos que

$$(\overline{-1}, \overline{-1}, ..., \overline{-1}) = \Psi(\overline{-1}) = \Psi(\overline{a}^{2^j m}) = (\overline{g_1}^{2^j a_1 m}, \overline{g_2}^{2^j a_2 m}, ..., \overline{g_k}^{2^j a_k m}). \tag{5.7}$$

Consequentemente essa igualdade será satisfeita se, e somente se,

$$p_i - 1 = 2^{r_i} m_i | 2^{j+1} a_i m \qquad p_i - 1 = 2^{r_i} m_i \nmid 2^j a_i m \tag{5.8}$$

para cada $i \in \{1, 2, ..., k\}$.

Observe que de (5.7) temos que

$$\begin{aligned} (\overline{g_i}^{2^j a_i m}) &\equiv -1 \bmod p_i \\ (\overline{g_i}^{2^j a_i m})^2 &\equiv (-1)^2 \bmod p_i \\ \overline{g_i}^{2^{j+1} a_i m} &\equiv 1 \bmod p_i \end{aligned}$$

Portanto a ordem de $\overline{g_i}$ é igual a $p_i - 1 = 2^{r_i} m_i$. Assim

$$\begin{aligned} 2^{j+1} a_i m &= 2^{r_i} m_i z \\ a_i &= 2^{r_i - j - 1} \cdot \frac{m_i}{(m, m_i)} \cdot \frac{m_i z}{[m, m_i]} \end{aligned}$$

com $z \in \mathbb{Z}$ e $\frac{m_i z}{[m, m_i]}$ ímpar.

Ressaltamos que $a_i \neq 2^{r_i - j} \frac{m_i q}{(m, m_i)}$. Portanto, 5.8 equivale a a_i ser múltiplo ímpar de $2^{r_i - j - 1} \frac{m}{(m, m_i)}$. Como $0 \leq a_i < p_i - 1$, então existem $2^j (m, m_i)$ opções para a_i. Portanto, o número de raízes para $X^{2^j m} + \overline{1}$ em $\mathbb{Z}_n$ é

$$2^{kj} (m, m_1)(m, m_2)...(m, m_k)$$

quando $j < min\{r_1, ..., r_k\}$ ou 0 caso contrário.

□

Com os resultados dos Lemas 5.6 e 5.7, podemos provar o Teorema 5.5.

Prova: Se n é pseudoprimo com respeito a a, então $a^{n-1} = \overline{1}$ por definição. Portanto, a possui inverso multiplicativo em $\mathbb{Z}_n$. Assim, devemos mostrar que vale a seguinte desigualdade

$$P_n = \frac{|\{a \in \mathbb{U}(n) : n \text{ é pseudoprimo com respeito a } a\}|}{n - 1} \leq \frac{1}{4}.$$

Caso 1: n não é livre de quadrados.

Por definição, $\exists$ primo p tal que $p^2 | n$. Seja t um natural satisfazendo $n = p^2 t$. Pelo Teorema 2.24, $\exists g \in \mathbb{N}$ tal que $0 < g < p^2$ e $\overline{g}$ é um gerador para $\mathbb{U}(p^2)$.

Assuma que n é pseudoprimo com respeito a $\overline{a}$, onde

$$a \in \mathbb{N}; 0 < a < n, \tag{5.9}$$

em particular,

$$a^{n-1} \equiv 1 \bmod n. \tag{5.10}$$

Vamos obter um limite superior para estes inteiros satisfazendo (5.9) e (5.10). De (5.10) temos que

$$\begin{aligned} a^{n-1} &\equiv 1 \bmod n \\ a^{n-1} - 1 &\equiv 0 \bmod p^2 t \\ a^{n-1} - 1 &\equiv 0 \bmod p^2 \\ a^{n-1} &\equiv 1 \bmod p^2. \end{aligned} \tag{5.11}$$

Como $\overline{g}$ é gerador de $\mathbb{U}(p^2)$, então

$$\exists i \in \mathbb{Z}; 0 \leq i < \phi(p^2) = p(p-1), a \equiv g^i \bmod p^2. \tag{5.12}$$

De (5.11) e (5.12), temos que

$$a^{n-1} \equiv g^{i(n-1)} \equiv 1 \bmod p^2.$$

Pelo Teorema 2.18, a ordem de $\overline{g}$ em $\mathbb{Z}_{p^2}$ divide $i(n-1)$. Isto é, $p(p-1)$ divide $i(p^2t-1)$. Logo, i tem que ser múltiplo de p. Portanto, $i = pj$, com $j \in \mathbb{Z}$ tal que $0 \leq j < p-1$.

De 5.9 e 5.12 temos que

$$a = g^{pj} + sp^2 \tag{5.13}$$

com $0 \leq s < t$ e $s \in \mathbb{Z}$.

Como j varia de 0 até $p-1$, e s varia de 0 até t, então temos no máximo $(p-1)t$ valores possíveis para a satisfazendo (5.13), com $0 < a < n$. Portanto n é pseudoprimo com respeito a no máximo $(p-1)t$ elementos de $\mathbb{Z}_n$. Então

$$P_n \leq \frac{(p-1)t}{n-1} = \frac{(p-1)t}{p^2t-1} < \frac{(p-1)t}{(p^2-1)t} = \frac{p-1}{(p+1)(p-1)} = \frac{1}{p+1} \leq \frac{1}{4}$$

pois n é ímpar, logo $p \geq 3$.

Caso 2: n é livre de quadrados.

Pelos Lemas 5.6 e 5.7, temos que n é pseudoprimo com respeito a exatamente

$$(m,m_1)(m,m_2)...(m,m_k) + \sum_{j=0}^{min\{r_1,r_2,...,r_k,r\}-1} 2^{kj}(m,m_1)(m,m_2)...(m,m_k)$$

elementos de $\mathbb{Z}_n$. Portanto

$$P_n = \frac{(m,m_1)...(m,m_k)}{n-1}\left(1 + \sum_{j=0}^{min\{r_1,...,r\}-1} 2^{kj}\right).$$

Substituindo a soma da progressão geométrica (P.G.) de razão 2^k pelo seu valor, temos que

$$P_n = \frac{(m,m_1)...(m,m_k)}{p_1p_2...p_k - 1}\left(1 + \frac{2^{min\{r_1,...,r\}k}-1}{2^k-1}\right).$$

Note que

$$\frac{1}{p_1...p_k-1} < \frac{1}{(p_1-1)(p_2-1)...(p_k-1)} = \frac{1}{2^{r_1+...+r_k}m_1m_2...m_k}.$$

Então

$$P_n < \frac{(m,m_1)...(m,m_k)}{2^{r_1+...+r_k}m_1...m_k}\left(1 + \frac{2^{min\{r_1,...,r\}k}-1}{2^k-1}\right).$$

Observe que

$$1 + \frac{2^{min\{r_1,...,r\}k}-1}{2^k-1} \leq \frac{2^{min\{r_1,...,r\}k}}{2^{k-1}},$$

assim, temos que

$$P_n < \left[\frac{2^{min\{r_1,...,r\}k}}{2^{r_1+...+r_k}}\right]\left[\frac{1}{2^{k-1}}\right]\prod_{i=1}^{k}\frac{(m,m_i)}{m_i}.$$

Agora, vamos dividir essa probabilidade nos casos possíveis, e mostraremos que em todos eles $P_n \leq \frac{1}{4}$.

Caso 2.1: Se $k \geq 3$.

Observe que nesse caso temos as seguintes desigualdades:

$$\frac{2^{min\{r_1,...,r\}k}}{2^{r_1+...+r_k}} \leq 1 \tag{5.14}$$

pois $min\{r_1,...,r\}k \leq r_1 + ... + r_k$.

$$\frac{1}{2^{k-1}} \leq \frac{1}{4} \tag{5.15}$$

$$\prod_{i=1}^{k}\frac{(m,m_i)}{m_i} \leq 1 \tag{5.16}$$

De (5.14), (5.15) e (5.16) concluímos que

$$P_n < \left[\frac{2^{min\{r_1,\ldots,r\}k}}{2^{r_1+\ldots+r_k}}\right]\left[\frac{1}{2^{k-1}}\right]\prod_{i=1}^{k}\frac{(m,m_i)}{m_i} \leq \frac{1}{4}.$$

Caso 2.2: Se $k = 2$ e $r_1 \neq r_2$, suponha que $r_1 < r_2$, então

$$\begin{aligned} P_n &< \left[\frac{2^{min\{r_1,r_2,r\}2}}{2^{r_1+r_2}}\right]\left[\frac{1}{2^{2-1}}\right]\prod_{i=1}^{2}\frac{(m,m_i)}{m_i} \\ &< \left[\frac{2^{2r_1}}{2^{r_1+r_2}}\right]\left[\frac{1}{2}\right]\prod_{i=1}^{2}\frac{(m,m_i)}{m_i} \end{aligned}$$

Observe que

$$\frac{2^{2r_1}}{2^{r_1+r_2}} = \frac{2^{2r_1}}{2^{r_1}2^{r_2}} = \frac{2^{r_1}}{2^{r_2}} \leq \frac{1}{2}$$

pois $r_1 < r_2$ e $r_i \in \mathbb{N}$. Portanto

$$P_n < \left[\frac{2^{r_1}}{2^{r_2}}\right]\left[\frac{1}{2}\right]\prod_{i=1}^{2}\frac{(m,m_i)}{m_i} \leq \frac{1}{4}.$$

Análogo para $r_1 > r_2$.

Caso 2.3: Se $k = 2$, $r_1 = r_2$ e $(m, m_1) \neq m_1$, então

$$P_n < \left[\frac{2^{2r_1}}{2^{2r_1}}\right]\left[\frac{1}{2}\right]\prod_{i=1}^{2}\frac{(m,m_i)}{m_i}.$$

Observe que

$$\prod_{i=1}^{2}\frac{(m,m_i)}{m_i} = \frac{(m,m_1)}{m_1}\cdot\frac{(m,m_2)}{m_2} \leq \frac{1}{2}$$

pois $(m, m_1) \neq m_1$. Portanto

$$P_n < 1 \cdot \left[\frac{1}{2}\right]\prod_{i=1}^{2}\frac{(m,m_i)}{m_i} \leq \frac{1}{4}.$$

Caso 2.4: Se $k = 2$, $r_1 = r_2$, $(m, m_1) = m_1$ e $(m, m_2) \neq m_2$, então

$$\prod_{i=1}^{2}\frac{(m,m_i)}{m_i} = \frac{(m,m_1)}{m_1}\cdot\frac{(m,m_2)}{m_2} = \frac{m_1}{m_1}\cdot\frac{(m,m_2)}{m_2} \leq \frac{1}{2}$$

pois $(m, m_2) \neq m_2$. Portanto,

$$P_n < 1 \cdot \left[\frac{1}{2}\right]\prod_{i=1}^{2}\frac{(m,m_i)}{m_i} \leq \frac{1}{4}.$$

Caso 2.5: Se $k = 2$, $r_1 = r_2$, $(m, m_1) = m_1$ e $(m, m_2) = m_2$.

Neste caso, temos que $n = p_1p_2$ e que $p_1 - 1$ e $p_2 - 1$ dividem $n - 1$. Podemos assumir que $p_1 > p_2$. Mas $n - 1 = p_1p_2 - 1 = (p_1 - 1)p_2 + p_2 - 1$ e daí $n - 1$ deixa resto $p_2 - 1 < p_1 - 1$ quando dividido por $p_1 - 1$. Um absurdo e o resultado segue.

□

Vejamos um exemplo utilizando o Algoritmo 5.4 para observarmos o que significa o Teorema 5.5. Vamos verificar se o número 983 é composto. Pelo **Passo 1**, temos que $983 - 1 = 2^1 \cdot 491$. Assim, verificamos que $k = 1$ e $q = 491$. Pelo **Passo 2**, tomando $b = 2$ temos que $2^{491} \equiv 1 \bmod 983$, ou seja, a saída será 'teste inconclusivo'. Isso significa que, pelo Teorema 5.5, o número 983 tem $\frac{1}{4}$ de chance de ser pseudoprimo, ou seja, tem $\frac{3}{4}$, ou 75% de chance de ser primo. Vamos fazer o teste agora para a base $b = 3$, ou seja, pelo **Passo 2** temos que $3^{491} \equiv 1 \bmod 983$, portanto a saída será 'teste inconclusivo'. Isso significa que, a chance desse número ser pseudoprimo é de $\frac{1}{4} \cdot \frac{1}{4} = \frac{1}{16}$, ou seja, esse número tem $\frac{15}{16}$, ou ainda $93,75\%$ de chance de ser um número primo. Vamos testar agora para a base $b = 24$, assim teremos que $24^{491} \equiv 1 \bmod 983$, como esta é a terceira base que estamos utilizando, então a chande desse número ser pseudoprimo é de $\frac{1}{4} \cdot \frac{1}{4} \cdot \frac{1}{4} = \frac{1}{64}$, ou seja, esse número tem $\frac{63}{64}$, ou ainda $98,4375\%$ de chance de ser um número primo. Para termos 100% de certeza que o número 983 é primo, precisaríamos fazer o teste para 25% das bases mais uma, porém observe que esse método é eficaz para poucas bases, calculando apenas para 3 bases distintas já obtivemos mais de 98% de certeza que o número 983 seja primo. A título de curiosidade, de fato o número 983 é primo.

5.3 CUSTO DOS MÉTODOS DE FATORAÇÃO E PRIMALIDADE

Vimos anteriormente que existem métodos para verificar se um número é primo ou composto. Veremos a seguir o custo assintótico para fazer a verificação dos métodos vistos neste trabalho.

5.3.1 CUSTO DO CRIVO DE ERATÓSTENES

O crivo de Eratóstenes consiste em testar se algum inteiro menor do que $\sqrt{n}$ é divisor de n.

Se n é primo, então o custo do Algoritmo 5.1 será $O(\log_b^2 n\sqrt{n}) = O(\log_b^2 nb^{\frac{1}{2}\log_b n})$. Como $\log_b n$ representa a quantidade de dígitos de n em sua representação na base b, e como estamos calculando o custo dos algoritmos em função do número de dígitos da entrada, esse algoritmo possui um custo não polinomial.

Se n é composto e p é o menor número primo que divide n, podemos notar

que este algoritmo terá um custo igual a $O(\log_b^2 n\frac{p}{2}) = O(\log_b^2 nb^{\log_b p})$, que também é um custo não polinomial.

Como exemplo, queremos verificar se um número n de $N = 150$ dígitos na base 10 é primo ou não. Note que $n \leq 10^N$, portanto, utilizaremos esse valor como n para obtermos o custo máximo do algoritmo. Para isso, devemos testar todos os primos até $\sqrt{10^{150}} = 10^{\frac{150}{2}}$. Pelo Teorema 5.2, precisaremos realizar $\pi\left(10^{\frac{150}{2}}\right)$ testes. Então o custo para testar o crivo será

$$\begin{aligned}
O\left(\pi\left(10^{\frac{150}{2}}\right)\right) &\sim O\left(\frac{10^{\frac{150}{2}}}{\log 10^{\frac{150}{2}}}\right) \\
&\sim O\left(\frac{1}{\log 10} \cdot \frac{10^{\frac{150}{2}}}{\frac{150}{2}}\right) \\
&\sim O\left(\frac{2}{\log 10} \cdot \frac{10^{\frac{150}{2}}}{150}\right) \\
&\sim O\left(\frac{1}{150} \cdot \sqrt{10^{150}}\right) \\
&\sim O\left(\frac{1}{N} \cdot \sqrt{10^{N}}\right) \\
&\sim O\left(\frac{\sqrt{n}}{\log n}\right)
\end{aligned}$$

pois, da relação entre N e n temos

$$\begin{aligned}
n &\sim 10^N \\
\log_{10} n &\sim N \\
N &\sim O\left(\log_{10} n\right) \\
&\sim O\left(\frac{\log n}{\log 10}\right) \\
&\sim O\left(\log n\right)
\end{aligned}$$

Se o computador demora 10^{-9} segundos para efetuar cada operação, então o tempo necessário para verificar se n é um número primo é de

$$10^{-9} \cdot \frac{\sqrt{n}}{\log n} \text{ segundos .}$$

Ou seja, o custo assintótico será

$$\begin{aligned}
10^{-9} \cdot \frac{\sqrt{n}}{\log n} &\sim 10^{-9} \cdot \frac{\sqrt{10^{150}}}{\log 10^{150}} \\
&\sim 6,6666 \cdot 10^{63} \text{ segundos.}
\end{aligned}$$

Podemos interpretar esse resultado de outras maneiras. Observe que o custo que encontramos é dado em segundos, se dividi-lo por 60, iremos saber o custo em minutos, no caso, aproximadamente $1,1111 \cdot 10^{62}$ minutos. Se dividi-lo por 60 novamente, iremos saber o custo em horas, no caso, aproximadamente $1,8518 \cdot 10^{60}$ horas. Se dividi-lo por 24, iremos saber o custo em dias, no caso, aproximadamente $7,7160 \cdot 10^{58}$ dias. Se dividi-lo por 30, iremos saber o custo em meses, no caso, aproximadamente $2,5720 \cdot 10^{57}$ meses. Se dividi-lo por 12, iremos saber o custo em anos, no caso, aproximadamente $2,1433 \cdot 10^{56}$ anos. Os cientistas afirmam que o Universo tem cerca de $13,7$ bilhões de anos, então se dividir o custo por $13,7$ bilhões, iremos encontrar o custo em "universos", no caso, aproximadamente $1,5645 \cdot 10^{46}$ "universos".

5.3.2 CUSTO DA FATORAÇÃO DE FERMAT

Vamos supor que $n = pq$ onde p e q são números primos. Desta forma podemos escrever $n = \left(\frac{p+q}{2} + \frac{p-q}{2}\right)\left(\frac{p+q}{2} - \frac{p-q}{2}\right) = (a+b)(a-b)$. Como n é ímpar, então a e b são inteiros.

Note que se $p = q$, então $n = \left(\frac{p+q}{2}\right)^2$ e o Algoritmo 5.3 para no primeiro laço do loop. Se $p \neq q$, colocamos $\frac{p-q}{2} = r$ (supondo, sem perda de generalidade, que $p > q$) e obtemos que $n + 1 + 3 + \ldots + (2r-1) = n + r^2 = pq + \left(\frac{p-q}{2}\right)^2 = \left(\frac{p+q}{2}\right)^2$. Isto significa que o algoritmo para no Passo r. Se considerarmos que verificar se o número $n + r^2$ é ou não um quadrado perfeito tem custo $O(\log^2(n+r^2)r) = O(\log^2(n+r^2)b^{\log r})$, este custo aumenta exponencialmente conforme cresce a diferença $p - q$.

Como exemplo, vamos verificar o custo para fatorar n de $N = 150$ dígitos na base 10. É fato que se um dos fatores primos de n for relativamente próximo de 0, o Crivo de Eratóstenes dará conta do recado. Outra observação é que o custo aumenta exponencialmente conforme cresce a diferença $p - q$. Por isso, vamos dividir o intervalo de 0 a n em 5 intervalos, sendo eles

$$\begin{array}{ll} \text{Intervalo 1} & (0, \sqrt[5]{n}\,] \\ \text{Intervalo 2} & \left(\sqrt[5]{n}, \sqrt[5]{n^2}\right] \\ \text{Intervalo 3} & \left(\sqrt[5]{n^2}, \sqrt[5]{n^3}\right] \\ \text{Intervalo 4} & \left(\sqrt[5]{n^3}, \sqrt[5]{n^4}\right] \\ \text{Intervalo 5} & \left(\sqrt[5]{n^4}, \sqrt[5]{n^5}\right) \end{array}$$

Vamos considerar que q pertença ao Intervalo 2 e p pertença ao Intervalo 4, assim, garantiremos que p, q não são muito próximos, e também que nenhum desses fatores são relativamente próximos de 0. Tomemos $q \sim \sqrt[5]{n^{\frac{3}{2}}} = n^{\frac{3/2}{5}} = n^{\frac{3}{10}}$ e $p \sim \sqrt[5]{n^{\frac{7}{2}}} = n^{\frac{7/2}{5}} = n^{\frac{7}{10}}$, então $n = pq$ é satisfeito, pois $n^{\frac{7}{10}} \cdot n^{\frac{3}{10}} = n$. Como $n \sim 10^{150}$, então $q \sim (10^{150})^{\frac{3}{10}} = 10^{45}$ e

$p \sim (10^{150})^{\frac{7}{10}} = 10^{105}$. Assim, $r = \left(\frac{10^{105}-10^{45}}{2}\right)$. Portanto, considerando que o computador demora 10^{-9} segundos para efetuar cada operação, o custo para fatorar n pela Fatoração de Fermat é de

$$O(\log^2(n+r^2)r) \sim 10^{-9} \cdot \left(\log^2\left(10^{150} + \left(\frac{10^{105}-10^{45}}{2}\right)^2\right) \cdot \left(\frac{10^{105}-10^{45}}{2}\right)\right) \quad (5.17)$$

Calculando (5.17), obtemos que o custo assintótico da Fatoração de Fermat, de um número de 150 dígitos, é de aproximadamente

$$\begin{array}{lll} & 2,1924 \cdot 10^{100} & \text{segundos} \\ \text{ou} & 3,6539 \cdot 10^{98} & \text{minutos} \\ \text{ou} & 6,0899 \cdot 10^{96} & \text{horas} \\ \text{ou} & 2,5375 \cdot 10^{95} & \text{dias} \\ \text{ou} & 8,4582 \cdot 10^{93} & \text{meses} \\ \text{ou} & 7,0485 \cdot 10^{92} & \text{anos} \\ \text{ou} & 5,1449 \cdot 10^{82} & \text{“universos”.} \end{array}$$

5.3.3 CUSTO DO TESTE DE MILLER

Segundo Henri Cohen (COHEN, 1996, pg. 422), o custo do Teste de Miller é essencialmente $O(ln^3n)$, entretanto, para algumas restrições este custo pode ser reduzido à $O(ln^2n)$.

Como exemplo, calcularemos o custo assintótico para verificar se um número n, de 150 algarismos, é primo. Seja $n \sim 10^{150}$. Se a máquina demora 10^{-9} segundos por operação, o custo assintótico será de

$$\begin{aligned} O(ln^3n) &\sim 10^{-9} \cdot \ln(10^{150}) \cdot \ln(10^{150}) \cdot \ln(10^{150}) \\ &\sim 10^{-9} \cdot 150^3 \cdot \ln 10 \cdot \ln 10 \cdot \ln 10 \\ &\sim 4,1202 \cdot 10^{-2} \\ &\sim 0,0412 \text{ segundos .} \end{aligned}$$

Ou seja, para verificar se um número é primo, o Teste de Miller é muito eficiente, pois ele nos diz se o número é primo em uma fração de segundo. Lembrando que para ter certeza que o número é primo de fato, temos que fazer o teste para mais bases diferentes, para cada vez mais a chance do número ser primo se aproximar de 100%. Isso obviamente vai elevar um pouco esse custo, dependendo da quantidade de bases que for selecionado para realizar o teste.

6 SEGURANÇA DO RSA

A segurança do RSA está diretamente ligada a fatoração do parâmetro n.

Se conhecemos um método eficiente para fatorar n, então quebramos o método RSA. Podemos nos perguntar se dado um método eficiente para quebrar o RSA, este método implicará em fatorar n? Como podemos quebrar o RSA?

Uma possível resposta é determinar um método para calcular $\phi(n) = (p-1)(q-1)$, não necessariamente passando pela fatoração de n.

Supondo que alguém conseguisse inventar uma forma de encontrar $\phi(n)$ a partir da chave (n, e) de codificação, na verdade esse alguém teria encontrado um algoritmo rápido de fatoração, pois sabemos que $\phi(n) = (p-1)(q-1)$, que no caso, seria conhecido. Iremos determinar p e q a partir disto. Observe que

$$\phi(n) = (p-1)(q-1) = pq - p - q + 1 = n - (p+q) + 1$$

ou seja, $p + q = n - \phi(n) + 1$. Note que

$$(p+q)^2 - 4n = (p^2 + 2pq + q^2) - 4pq = p^2 - 2pq + q^2 = (p-q)^2$$

portanto $p - q = \sqrt{(p+q)^2 - 4n}$. Assim, conhecendo $p+q$ e $p-q$ calculamos facilmente p e q, isto é, fatoramos n. Portanto, não adianta calcular $\phi(n)$ sem fatorar n.

Diante desta situação, acredita-se que quebrar o RSA e fatorar n sejam problemas equivalentes, embora até agora isto não tenha sido provado (COUTINHO, 2009, pg. 187).

6.1 CUSTO DE IMPLEMENTAÇÃO

Vimos que o custo do Teste de Miller para encontrar números primos é relativamente baixo. Também é notável que o custo para implementar o RSA é relativamente baixo, já que se trata de calcular congruências de potências envolvendo o Algoritmo 4.9, que tem um custo máximo de $O(\ln^3 n)$.

Portanto, é possível implementar o RSA sem grandes dificuldades.

6.2 CUSTO DE FATORAÇÃO

Vimos anteriormente que, para quebrar o método RSA, a princípio, devemos fatorar n. Neste trabalho, estudamos dois métodos de fatoração, o Crivo de Eratóstenes,

descrito no Algoritmo 5.1 e a Fatoração de Fermat, descrito no Algoritmo 5.3.

Calculamos o custo assintótico de ambos os métodos de fatoração, e nos dois casos, esse custo não é polinomial. Calculamos também, como exemplo, o custo para fatorar um número com 150 dígitos, e o resultado foi um tempo de "universos", ou seja, com esses métodos conhecidos, é praticamente inviável fatorar um número relativamente grande.

6.3 CUIDADOS

A partir dos métodos de fatoração estudados, notamos que, se um dos fatores de n for um primo próximo de zero, pelo Algoritmo 5.1, é possível fatorá-lo com um custo bem pequeno. Portanto, é importante que os primos p, q escolhidos sejam ambos relativamente grandes. Outra observação, é que, se p e q forem números próximos um do outro, pelo Algoritmo 5.3 é possível fatorar n de forma rápida, pois o custo desse algoritmo cresce exponencialmente a medida que a distância entre p e q aumenta. Portanto, é importante que p e q sejam distantes um do outro.

Assim, concluímos que p e q não podem ser primos próximos, e nenhum deles pode ser relativamente pequenos, caso contrário, a segurança do método não é garantida.

7 CONSIDERAÇÕES FINAIS

Neste trabalho é estudado conceitos de aritmética modular, assim como algoritmos fundamentais, como o algoritmo da divisão de Euclides, que é base para a aritmética modular. Além disso, também estudamos conceitos de grupos, como classes de congruências, ordem e gerador de um grupo, apesar de não abordar com afinco as propriedades de grupos que, conforme algumas especificidades, podem ser chamados de anéis ou corpos.

Também é visto os tipos de criptografia, que é o método de chave pública e o método de chave privada, e abordamos com afinco, além de apresentar um exemplo de implementação, o método RSA, que por sua vez, está relacionado ao cálculo de congruências envolvendo restos de potências. O grande foco do trabalho foi estudar o RSA, entendê-lo, e saber por que funciona e o que garante sua segurança.

Para verificar a segurança do RSA, precisamos estudar sobre algoritmos de fatoração e algoritmos para encontrar números primos, no caso o Crivo de Eratóstenes, Fatoração de Fermat e Teste de Miller, este último, é um teste probabilístico para encontrar números primos. Além disso, também calculamos o custo assintótico de cada algoritmo, além de calcular para um número $n \sim 10^{150}$ para comparar os resultados práticos desses algoritmos. Para isso, precisamos buscar a definição de custo assintótico, e estudamos também o custo das operações básicas.

Por fim, conseguimos compreender a segurança do método, que está baseada na impraticabilidade de fatorar números suficientemente grandes. Também conseguimos compreender os cuidados que temos que tomar na hora de escolher os parâmetros, para que o método seja realmente seguro, que é manter uma distância considerável entre os números primos escolhidos, além de assegurar que nenhum deles será relativamente pequeno.

REFERÊNCIAS

COHEN, H. *A course in computational algebraic number theory*. [S.l.]: Springer, 1996. v. 3.

COUTINHO, S. C. *Números inteiros e criptografia RSA*. [S.l.]: Rio de Janeiro: Instituto Nacional de Matemática Pura e Aplicada, 2009.

COUTINHO, S. C. Criptografia. *Rio de Janeiro: IMPA/SBM, Programa de Iniciação Científica da OBMEP*, 2015.

HEFEZ, A. *Elementos de Aritmética*. [S.l.]: Sociedade Brasileira de Matemática, 2011.

HOFFSTEIN, J.; PIPHER, J. C.; SILVERMAN, J. H.; SILVERMAN, J. H. *An introduction to mathematical cryptography*. [S.l.]: Cham: Springer, 2008. v. 1.

LEMOS, M. *Criptografia, números primos e algoritmos*. [S.l.]: Rio de Janeiro: IMPA, 2010.

RIBENBOIM, P. *Números primos: mistérios e recordes*. [S.l.]: Rio de Janeiro: IMPA, 2001.

SANTANA, A. G. d. Criptografia de curvas elípticas sobre extensões de corpos finitos. *Londrina: Dissertação (Mestrado em Matemática Aplicada e Computacional) – Universidade Estadual de Londrina*, 2013.

SANTANA, A. G. d.; SILVA, A. L. d. Teoria dos números na criptografia rsa. *Londrina: Universidade Estadual de Londrina*, 2008.