Criptografia RSA Gaussiana

Luis Antonio Coêlho

Trabalho de Conclusão de Curso - apresentado à
Faculdade de Tecnologia da
Universidade Estadual de Campinas

Orientadora: **Profa. Dra. Juliana Bueno-Soler**

14 de junho de 2017

AGRADECIMENTOS

Agradeço neste trabalho primeiramente a Deus que permitiu que tudo isso acontecesse, ao longo de minha vida, e não somente nestes anos como universitário, mas que em todos os momentos é o maior mestre que alguém pode conhecer.Agradeço a todos os professores por me proporcionar o conhecimento não apenas racional, mas a manifestação do caráter e afetividade da educação no processo de formação profissional, por tanto que dedicaram a mim, não somente por terem me ensinado, mas por terem me feito aprender. A palavra mestre, nunca fará justiça aos professores dedicados aos quais sem nominar terão os meus eternos agradecimentos. Agradeço a minha mãe Raquel, heroína que me deu apoio, incentivo nas horas difíceis, de desânimo e cansaço e a todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

RESUMO

A motivação para esta monografia é analisar a viabilidade de uma criptografia RSA baseada em números primos de Gauss, isto é, números primos definidos dentro de um subconjunto do corpo dos complexos, os chamados *inteiros de Gauss*. Para iniciar esse estudo fazemos um levantamento dos principais resultados de teoria de números necessários para a compreensão do algoritmo RSA clássico. Como o leitor irá notar a aritmética modular é central nessa construção na medida em que resultados como o *Teorema de Fermat* e o *Teorema Chinês do Resto* são elementos centrais da criptografia RSA.

Após uma exposição detalhada do método criptográfico RSA clássico iniciamos uma discussão sobre a criptografia RSA Gaussiana, tema desta monografia. O percurso escolhido para essa exposição será o mesmo apresentado no caso clássico tentando identificar os pontos frágeis para essa contrução. Finalmente fazemos uma discussão sobre os desafios dentro deste novo campo para a computação.

SUMÁRIO

INTRODUÇÃO

O sigilo sempre foi uma arma explorada pelos seres humanos para vencer certas batalhas, e até mesmo para a cotidiana missão de se comunicar. Foi a partir dessa necessidade que se criou a *criptografia*, nome dado ao conjunto de técnicas usadas para se comunicar em códigos. Seu objetivo é garantir que apenas os envolvidos na comunicação possam compreender a mensagem codificada (ou criptogtafada), garantindo que terceiros não saibam o que foi conversado.

Para compreender como funciona o processo de codificação e decodificação faz-se necessário o uso de uma série de termos técnicos, e para fins pedagógicos iremos introduzir tais conceitos apresentando um dos primeiros algoritmos criptográficos que se tem conhecimento, a criptografia de César. Para mais detalhes sobre o tema sugerimos que veja Criptografia, por Coutinho[Cou07].

A chamada *criptografia de César*, criada pelo imperador romano César Augusto, consistia em substituir cada letra da mensagem por outra que estivesse a três posições a frente, como, por exemplo, a letra **A** que neste algoritmo é substituída pela letra **D**.

Uma forma muito natural de se generalizar o algoritmo de César é fazer a troca de cada letra da mensagem por outra que venha em uma posição qualquer fixada. A chamada *criptografia de substituição monoalfabética* consiste em substituir cada letra por outra que ocupe n posições à sua frente, sendo que o número n é conhecido apenas pelo emissor e pelo receptor da mensagem. O número n é a *chave criptográfica*. Para decifrar a mensagem, precisamos substituir as letras que formam a mensagem criptografada pelas letras que estão n posições antes.

O algoritmo monoalfabético tem a característica indesejada de ser de fácil decodificação, pois possui apenas 26 chaves possíveis, e isso faz com que no máximo em 26 tentativas o código seja decifrado. Com o intuito de dificultar a quebra do código monoalfabético foram propostas as *cifras de substituição polialfabéticas* em que a chave criptográfica passa a ser uma *palavra* ao invés

de um número. A ideia é usar as posições ocupadas pelas letras da chave para determinar o número de posições que devemos avançar para obter a posição da letra encriptada. Vejamos, por meio de um exemplo, como funciona esse sistema criptográfico.

Sejam "SENHA" a nossa chave criptográfica e "ABOBORA" a mensagem a ser encriptada. Abaixo colocamos as letras do alfabeto com suas respectivas posições. Observe que repetimos a primeira linha de letras para facilitar a localização da posição da letra encriptada e usamos a barra para indicar que estamos no segundo ciclo.

1	2	3	4	5	6	7	8	9	10	11	12	13
A	B	C	D	**E**	F	G	**H**	I	J	K	L	M

14	15	16	17	18	19	20	21	22	23	24	25	26
N	O	P	Q	R	**S**	T	U	V	X	Y	W	Z

27	28	29	30	31	32	33	34	35	36	37	38	39
$\overline{A}$	$\overline{B}$	$\overline{C}$	$\overline{D}$	$\overline{E}$	$\overline{F}$	$\overline{G}$	$\overline{H}$	$\overline{I}$	$\overline{J}$	$\overline{K}$	$\overline{L}$	$\overline{M}$

Vejamos como encriptar a palavra "ABOBORA". Iniciamos o processo escrevendo a mensagem. Ao lado de cada letra da mensagem aparece entre parênteses o número que indica a sua posição. Abaixo da mensagem escrevemos as letras da chave criptográfica, repetindo-as de forma cíclica quando necessário. Analogamente, ao lado de cada letra da chave aparece entre parênteses o número da posição ocupada de cada letra, e o sinal de soma indica que devemos avançar aquele número de posições. Ao final do processo aparecem as letras encriptadas. Entre parênteses está a posição resultante da combinação das posições da mensagem e da chave.

$A(1)$	$B(2)$	$O(15)$	$B(2)$	$O(15)$	$R(18)$	$A(1)$	Mensagem
$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	
$S(+19)$	$E(+5)$	$N(+14)$	$H(+8)$	$A(+1)$	$S(+19)$	$E(+5)$	Chave
$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	
$T(20)$	$G(7)$	$C(29)$	$J(10)$	$P(16)$	$K(37)$	$F(6)$	Mensagem encriptada

Observe que a encriptação polialfabética é mais difícil de ser quebrada que a monoalfabética uma vez que letras iguais não têm, necessariamente, a mesma encriptação. Neste tipo de criptografia o emissor precisa passar a chave para o receptor da mensagem de forma segura para que o receptor

possa decifrar a mensagem, isto é, a chave usada para encriptar a mensagem é a mesma que deve ser usada para decifrar a mensagem. Veremos que esse é justamente o ponto fraco neste tipo de encriptação pois usa a chamada *chave simétrica*, ou seja, a chave usada pelo emissor para codificar a mensagem é a mesma usada pelo receptor para decodificar a mensagem. Nesse processo, a chave deve ser mantida em segredo e bem guardada para garantir que o código não seja quebrado, e isso requer algum tipo de contato físico entre emissor e receptor da mensagem.

Durante a Primeira Guerra Mundial o contato físico para a troca de chaves era complicado, e isso estimulou a criação de máquinas automáticas de criptografia. O *Enigma* foi uma dessas máquinas e era utilizada pelos alemães tanto para criptografar como para descriptografar códigos de guerra. Semelhante a uma máquina de escrever, os primeiros modelos foram patenteados por Arthur Scherbius em 1918. Essas máquinas ganharam popularidade entre as forças militares alemãs devido à facilidade de uso e sua suposta indecifrabilidade do código.

O matemático Alan Turing foi o responsável por quebrar o código dos alemães durante a Segunda Guerra Mundial. A descoberta de Turing mostrou a fragilidade da criptografia baseada em chave simétrica e colocou novos desafios à criptografia. O grande problema passou a ser a questão dos protocolos, isto é, como transmitir a chave para o receptor de forma segura sem que seja necessário o contato físico entre as partes.

Em 1949, com a publicação do artigo *Communication Theory of Secrecy Systems* [Sha49] de Shannon, temos a inauguração da criptografia moderna. Neste artigo ele escreve matematicamente que cifras teoricamente inquebráveis são semelhantes às cifras polialfabéticas. Com isso ele transformou a criptografia que até então era uma arte, em uma ciência.

Em 1976 Diffie e Hellman publicaram *New Directions in Cryptography* [DH76]. Neste artigo há a introdução ao conceito de *chave assimétrica*, onde há chaves diferentes entre o emissor da mensagem e seu receptor. Com a assimetria de chaves não era mais necessário um contato tão próximo entre emissor e receptor. Neste mesmo artigo é apresentado o primeiro algoritmo de criptografia de chave assimétrica ou como é mais conhecido nos dias atuais *Algoritmo de Criptografia de Chave Pública*, o protocolo de Diffie-Hellman.

Um dos algoritmos mais famosos da criptografia de chave pública é o *RSA* [RSA78], algoritmo desenvolvido por Rivest, Shamir e Adleman. Este algoritmo se tornou popular por estar presente em muitas aplicações de alta segurança, como bancos, sistemas militares e servidores de internet.

Para que se possa compreender por completo o algoritmo faz-se necessário possuir alguns conhecimentos em teoria de números como fatoração e aritmética modular. Estes conhecimentos serão apresentados mais adiante neste trabalho.

No algoritmo RSA existe uma chave pública n, que é a multiplicação dos primos p e q. O emissor E codifica a mensagem usando um número primo p. Em seguida E envia publicamente a mensagem codificada junto com a chave n para o receptor R. R possui o número q, que juntamente ao número n servem para decodificar a mensagem.

Embora a quebra do RSA seja aparentemente simples, bastando fatorar n para descobrir seus fatores, o grande problema é na realidade computacional, pois usa-se como p e q números primos muito altos, próximos a 2^{512}. Com um número tão alto um computador comum levaria bem mais que uma vida humana para decifrar a mensagem.

Com base nestes conhecimentos sobre criptografia, temos que o objetivo deste trabalho é analisar a viabilidade de uma criptografia inspirada pelo algoritmo RSA clássico, a qual substitui os números primos pelo conjunto denominado de *primos de Gauss* [PR13], resultando, assim, no que chamamos por *criptografia RSA gaussiana*. Para que tal algoritmo seja viável é necessário adaptar uma série de resultados relativos aos números primos aos número primos de Gauss. Dessa forma, nossa tarefa será adaptar tanto quanto o possível os primos de Gauss às demonstações desses teoremas.

Como se trata de uma proposta inovadora, deixamos para trabalhos futuros uma análise comparativa entre as criptografias RSA clássica e a RSA gaussiana.

1. UM PASSEIO PELA TEORIA DE NÚMEROS

A teoria de números é umas das mais antigas áreas da matemática e dedica-se ao estudo relacionado a propriedades relativas aos números inteiros tais como: a questão da fatoração, máximo divisor comum, primalidade, entre outras. Ao longo deste capítulo mostraremos os principais resultados de teoria de números essenciais para a compreensão do método de criptografia de chave pública conhecido como RSA.

1.1 Números Primos e Fatoração Única

Os números primos ocupam lugar importante tanto na teoria de números quanto na criptografia RSA: na primeira por serem capazes de gerar todos os elementos do conjunto dos números inteiros e suas consequentes propriedades; na segunda pelo fato de formarem um conjunto infinito e permiir com isso que se tome um primo de dimensão estrondosa para codificar uma mensagem, consequentemente dificultando que um código seja decifrado por terceiros em tempo razoável, como mostraremos ao longo desta monografia.

Os primos atuam como átomos dentro do conjunto dos números inteiros no sentido em que todo número pode ser escrito de forma única como um produto de primos. Esse fato é consequência do chamado *Teorema da Fatoração Única* também conhecido por *Teorema Fundamental da Aritmética*. Além de ser um resultado fundamental para a teoria de números, ele também é um dos pilares da criptografia RSA, pois a decodificação de uma mensagem vai passar pela fatoração de um número. Para demonstrar esse teorema é preciso ter a disposição o seguinte teorema.

Teorema 1.1.1 (Teorema de divisão)**.** *Sejam a e b inteiros positivos. Existem números inteiros q (quociente) e r (resto) tais que:*

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

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

Demonstração. Confira em [Cou14], seção 3 do capítulo 1, p. 22. □

O teorema acima faz duas afirmações: a primeira que o quociente e o resto da divisão sempre existem; a segunda, que o quociente e o resto são únicos. A garantia da unicidade é o ponto crucial na aplicação à criptografia RSA, pois assim temos a garantia de que uma mensagem possa ser decodificada de maneira única. Um outro resultado igualmente importante é o *Algoritmo de Euclides* mais conhecido como método para se calcular o máximo divisor comum entre dois números. Esse resultado é importante para definir o que entendemos por números primos e consequentemente para o teorema da fatoração única, o qual mostra como expressar um número em fatores primos de forma única.

Descrição do *algoritmo euclidiano*:

- **Entrada - etapa 1:** divida a por b e ache o resto r_1;
  - Se $r_1 = 0$, escreva "$mdc(a, b) = 1$" e pare o processo.
  - Se $r_1 \neq 0$, volte para a entrada.
- **Entrada - etapa 2:** divida b por r_1 e ache o resto r_2;
  - Se $r_2 = 0$, escreva $mdc(a, b) = r_1$ e pare o processo.
  - Se $r_2 \neq 0$, volte para a entrada.
- **Entrada - etapa 3:** divida r_1 por r_2 e ache o resto r_3;
  - Se $r_3 = 0$, escreva $mdc(a, b) = r_2$ e pare o processo.
  - Se $r_3 \neq 0$, volte para a entrada.
- E assim por diante.

O algoritmo acima produz a seguinte sequência de divisões, e o processo segue até que se obtenha um resto igual a zero, nesse caso o procedimento termina:

$$\begin{array}{rcll} a & = & bq_1 + r_1 & \text{e} \quad 0 \leq r_1 < b \\ b & = & r_1q_2 + r_2 & \text{e} \quad 0 \leq r_2 < r_1 \\ r_1 & = & r_2q_3 + r_3 & \text{e} \quad 0 \leq r_3 < r_2 \\ r_2 & = & r_3q_4 + r_4 & \text{e} \quad 0 \leq r_4 < r_3 \\ & \vdots & & \\ r_{k-2} & = & r_{k-1}q_k + r_k & \text{e} \quad 0 \leq r_k < r_{k-1} \\ r_{k-1} & = & r_kq_{k+1} + 0 & \text{e} \quad 0 \leq 0 < r_k \end{array}$$

As informações relevantes obtidas no procedimento podem ser organizado em uma tabela do seguinte modo:

a	b	r_1	r_2	$\cdots$	r_{k-1}	r_k
r_1	r_2	r_3	r_4		0	

A construção da tabela se inicia com a inserção dos números a e b na linha de cima e em seguida preenchemos a o primeiro bloco da linha de baixo com o valor do resto da divisão de a por b. Se o resto for diferente de zero, inserimos r_1 no terceiro bloco da linha de cima e fazemos a divisão de b por r_1 e escrevemos r_2 no segundo bloco da linha de baixo. O processo termina quando obtiver um zero na linha de baixo da tabela e o máximo divisor comum será o último valor inserido na primeira linha da tabela antes do processo terminar.

Vejamos como exemplo, como obter o máximo divisor comum entre 1234 e 54 usando o algoritmo acima.

1234	54	46	8	6	2
46	8	6	2	0	

Para a criptografia RSA será conveniente trabalhar com uma versão estendida do método descrito acima o qual chamamos *algoritmo euclidiano estendido*. Na versão estendida vão aparecer dois números que serão importantes para obter inverso multiplicativo na aritmética modular, como veremos na próxima seção.

Teorema 1.1.2 (Algoritmo Euclidiano Estendido)**.** *Sejam a e b inteiros positivos e seja d o máximo divisor comum entre a e b. Existem inteiros α e β tais que:*

$$\alpha \cdot a + \beta \cdot b = d$$

Diferente do teorema de divisão, o algoritmo euclidiano estendido não garante a unicidade com relação aos inteiros α e β, mas veremos que esse problema acaba sendo contornado por termos à disposição um método eficiente para calcular esses números. A versão do algoritimo que vamos apresentar é devida a Donald Knuth, um grande nome para a computação, conforme afirma Coutinho em [Cou14].

Vejamos como o método estendido pode ser descrito:

$$\begin{array}{rcll} a & = & bq_1 + r_1 & \text{e} \quad r_1 = ax_1 + by_1 \\ b & = & r_1q_2 + r_2 & \text{e} \quad r_2 = ax_2 + by_2 \\ r_1 & = & r_2q_3 + r_3 & \text{e} \quad r_3 = ax_3 + by_3 \\ r_2 & = & r_3q_4 + r_4 & \text{e} \quad r_4 = ax_4 + by_4 \\ & \vdots & & \vdots \\ r_{k-3} & = & r_{k-2}q_{k-1} + r_{k-1} & \text{e} \quad r_{k-1} = ax_{k-1} + by_{k-1} \\ r_{k-2} & = & r_{k-1}q_k + 0 & \text{e} \quad r_k = 0 \end{array}$$

Os números $x_1, x_2, \cdots, x_{k-1}$ e $y_1, y_2, \cdots, y_{k-1}$ são inteiros a serem determinados durante o processo pelas seguintes expressões:

$$x_j = x_{j-2} - q_j x_{j-1} \quad \text{e} \quad y_j = y_{j-2} - q_j y_{j-1}$$

As informações obtidas acima podem ser inseridas numa tabela da seguinte forma:

Restos	Quocientes	x	y
a	$*$	x_{-1}	y_{-1}
b	$*$	x_0	y_0
r_1	q_1	x_1	y_1
r_2	q_2	x_2	y_2
r_3	q_3	x_3	y_3
$\vdots$	$\vdots$	$\vdots$	$\vdots$
r_{k-2}	q_{k-2}	x_{k-2}	y_{k-2}
0	q_{k-1}	$*$	$*$

Para fins práticos iniciamos o processo fixando os seguintes valores para as variáveis inciais que não aparecem no processo: $x_{-1} = y_0 = 1$ e $x_0 = y_{-1} = 0$. Com isso o processo se inicia e uma sequência de divisões é gerada até que se obtenha resto zero, finalizando o processo. A partir daí os valores $\alpha = x_{k-2}$ e $\beta = y_{k-2}$ que aparecem no enunciado do teorema são determinados. Para detalhes acerca da demonstração do método descrito acima recomendamos a seção 6 do capítulo 1 de [Cou14].

Vejamos o exemplo de como calcular o máximo divisor comum de 1234 e 54 usando o algoritmo estendido:

Restos	Quocientes	x	y
1234	$*$	1	0
54	$*$	0	1
46	22	$x_1 = 1 - 22 \cdot 0 = 1$	$y_1 = 0 - 22 \cdot 1 = -22$
8	1	$x_2 = 0 - 1 \cdot 1 = -1$	$y_2 = 1 - 1(-22) = 23$
6	5	$x_3 = 1 - 5(-1) = 6$	$y_3 = -22 - 5 \cdot 23 = -137$
2	1	$x_4 = -1 - 1 \cdot 6 = -7$	$y_4 = 23 - 1 \cdot (-137) = 160$
0	3	$*$	$*$

Daí, temos que $\alpha = -7$ e $\beta = 160$ e pelo Teorema 1.1.2 segue que $mdc(1234, 54) = (-7) \cdot (1234) + (160) \cdot (54) = 2$, como esperávamos.

Como dissemos, a vantagem em usar o algoritmo acima está no fato de termos um procedimento para determinar os números α e β, que como veremos serão úteis para o RSA.

Agora, estamos em condições de definir o que entendemos por números primos para então atingir nossa meta com este capítulo: o teorema da fatoração única.

Definição 1.1.3. Um número inteiro p é *primo* se $p \neq \pm 1$ e os únicos divisores de p são ± 1 e $\pm p$.

São exemplos de números primos: ± 2, ± 3, ± 5, ± 7, ± 11, ± 13, etc.

Um número inteiro, diferente de ± 1, que não é primo é chamado *composto*. Observe que os números 1 e -1 não são nem primos e nem compostos. A exclusão desses números do conjunto dos primos tem a finalidade de garantir a unicidade da fatoração no teorema a seguir. Um outro aspecto a se destacar acerca desse par de números é que eles são os únicos em $\mathbb{Z}$ que admitem inverso multiplicativo. Falaremos mais sobre esse assundo mais adiante.

Teorema 1.1.4 (Teorema da Fatoração Única)**.** *Dado um inteiro positivo $n \geq 2$ podemos sempre escrevê-lo, de modo único, na forma*

$$n = p_1^{e_1} \times p_2^{e_2} \times \cdots \times p_k^{e_k}$$

onde $1 < p_1 < p_2 < p_3 < \cdots < p_k$ são números primos e $e_1, \cdots, e_k$ são inteiros positivos.

Demonstração. Veja capítulo 2 em [Cou14]. □

No teorema acima, os expoentes e_i, para $1 \leq i \leq k$ são chamados de *multiplicidades*, pois indicam a quantidade de vezes que um mesmo número primo ocorre na fatoração. A prova de que é sempre possível encontrar os fatores usados para decompor o número em fatores primos consiste no procedimento para fatorar um número, esse procedimento é chamado *Algoritmo de Euclides*: trata-se do método que se aprende na escola para fatorar um número e que não iremos detalhar aqui. Como bem sabemos, esse método é bastante ineficaz quando pensamos em números muito grande, pois pode requerer que se realize uma sequência bem grande de divisões. A prova garante que o procedimento termina, mas o que se nota é que tal procedimento é muito ineficiente no sentido em que demanda muito tempo para se chegar a uma resposta dependendo do número que desejamos fatorar. Na literatura existem vários algoritmos de fatoração que tornam o método mais eficiente, no entanto nenhum desses métodos funciona bem para todos os números inteiros. A criptografia RSA aproveita a ineficiência dos métodos para fatorar um número para garantir a segurança do seu sistema. É um problema em aberto saber se existe ou não um método rápido para fatorar números inteiros.

A demonstração do teorema acima requer uma série de resultados acerca de números primos os quais detalharemos abaixo. O interesse em apresentar as demonstrações de tais resultados é devido ao fato de estarmos interessados em adaptar tais provas para os primos de Gauss, cujo objetivo é compreender as dificuldades para implementar um método de criptografia RSA baseado em primos de Gauss.

Teorema 1.1.5. *Sejam a e b inteiros positivos e suponhamos que a e b são primos entre si.*

1. *Se b divide o produto $a \cdot c$ então b divide c.*
2. *Se a e b dividem c então o produto $a \cdot b$ divide c.*

Demonstração. 1. Se a e b são primos entre si, então o máximo divisor comum entre a e b é 1, isto é, $mdc(a, b) = 1$. Pelo algoritmo euclidiano estendido (Teorema 1.1.2), temos que existem inteiros α e β tais que $\alpha \cdot a + \beta \cdot b = 1$. Então, multiplicando toda a expressão por c temos que:

$$\alpha \cdot a \cdot c + \beta \cdot b \cdot c = c \tag{1.1}$$

Dado que b divide $a \cdot c$ e b divide $c \cdot b$, então b divide $\alpha \cdot ac + \beta \cdot bc$. Portanto, a partir da igualdade (1.1) temos que b divide c.

2. Se a divide c, então existe $t \in \mathbb{Z}$ tal que $c = a \cdot t$. Como, por hipótese, b divide c e a e b são primos entre si, então b tem que dividir t. Logo, para algum t vale que $t = b \cdot k$. Portanto, $c = a \cdot t = a(b \cdot k) = (a \cdot b)k$ é divisível por $a \cdot b$.

□

Teorema 1.1.6 (Propriedade Fundamental dos Primos)**.** *Seja p um número primo e sejam a e b inteiros positivos. Se p divide o produto $a \cdot b$ então p divide a ou p divide b.*

Demonstração. Se a e p não forem primos entre si então o máximo divisor comum entre eles é p, logo p divide a. Suponhamos que a e p são primos entre si, isto é, $mdc(p, a) = 1$. Como, por hipótese, p divide $a \cdot b$, então pelo Teorema 1.1.5 segue que p divide b. □

Estamos interessados, agora, em mostrar que a lista de primos é infinita, para tal vejamos o seguinte resultado intermediário.

Teorema 1.1.7 (Existência de Divisor Primo)**.** *Se n é um número inteiro positivo composto, então n tem um divisor primo p tal que $p \leq \sqrt{n}$.*

Demonstração. Se n é um número composto e positivo, podemos supor que $n = a \cdot b$, com $1 < a \leq b$.

1. De $1 < a$, temos que existe um primo p que divide a (Teorema 1.1.4), com $p \leq a$, daí $p^2 \leq a^2$.

2. De $a \leq b$ temos que $a^2 \leq a \cdot b = n$

De (1) e (2) segue que $p^2 \leq n$, logo $p \leq \sqrt{n}$. □

Teorema 1.1.8 (Infinidade de Primos)**.** *Existe uma quantidade infinita de números primos.*

Demonstração. Suponha, por redução ao absurdo, que existe apenas uma quantidade finita de números primos: $p_1, p_2, \cdots p_n$. Tome $a = 1 + p_1 \times p_2 \times \cdots \times p_n$ um número inteiro. Claramente $a > p_i$ para cada $1 \leq i \leq n$, logo a deve ser um número composto, caso contrário a lista acima estaria incompleta. Como a é composto, pelo Teorema 1.1.7 existe um primo p_i tal que divide a. Dado que p_i divide $p_1 \times p_2 \times \cdots \times p_n$ e divide a, então p_1 divide 1. Absurdo, pois o único divisor de 1 é ele mesmo. Portanto, é falso supor que a lista de primos seja finita, logo ela deve ser infinita. □

Existe, ainda, um outro debate acerca de como gerar os números primos. Existem diversos métodos para gerar os primos, como o Crivo de Eratóstenes, o mais antigo deles, mas não envolve nenhuma fórmula específica. No entanto, todos esses métodos mostram-se ineficazes, como mostra Coutinho em [Cou14].

Como vimos, o Teorema 1.1.4 garante que um número pode ser decomposto em fatores primos de forma única e o Teorema 1.1.8 garante a existência de uma infinidade de números primos, no entanto, os procedimentos atrelados a esses resultados são todos muito ineficientes em termos computacionais. Para implementar a criptografia RSA, como iremos trabalhar com números muito grandes, vamos precisar denotá-los por meio de potências e isso requer métodos eficazes para trabalhar com esses números, por essa razão vamos precisar da aritmética modular.

1.2 Aritmética Modular

Para compreender a intuição por trás da aritmética modular é interessante pensar na ideia de *ciclicidade*, isto é, em fatos que ocorrem após um determinado período constante ou ciclo. Por exemplo, o nascer do sol é um evento que permite marcar um ciclo, pois ocorre sempre após um ciclo de 24 horas; a data de seu aniversário é outro evento que permite marcar um ciclo, pois ocorre a cada 12 meses; e assim por diante. Trabalhar com objetos que têm um corportamento cíclico requer que tenhamos uma nova forma de operar, pois quando somamos 13 com 15 o resultado pode ser 4 se estivermos pensando em termos de horas. Quando termina um ciclo de 24 horas, nesse mesmo instante inicia-se um novo ciclo (do dia seguinte). A repetição de uma mesma hora indica o completamento de um ciclo (24 horas) a partir do ponto estabelecido como marco inicial.

Quando mostramos o processo de codificação e de decodificação de um código em *cifras de substituição polialfabéticas* você deve ter notado que precisamos repetir o alfabeto a fim de podermos operar com as posições ocupadas por uma determinada letra do alfabeto. A repetição do alfabeto foi usada para mostrar as diferentes posições ocupadas por uma mesma letra. Observe que há nesse processo o estabelecimento de um ciclo determinado pela quantidade de letras do alfabeto.

A partir dos ciclos podemos construir classes de números representando as possíveis marcações (inteiras). Por exemplo, no caso das horas temos as

seguintes classes:

- $0h = \{x \in \mathbb{N} : x = 0 + 24\} = \{0, 24, 48, 72, \cdots\}$
- $1h = \{x \in \mathbb{N} : x = 1 + 24\} = \{1, 25, 49, 73, \cdots\}$
- $2h = \{x \in \mathbb{N} : x = 2 + 24\} = \{2, 26, 50, 74, \cdots\}$
- $\vdots$
- $23h = \{x \in \mathbb{N} : x = 23 + 24\} = \{23, 47, 71, 95, \cdots\}$

No exemplo das posiçẽes ocupadas pelas letras do alfabeto temos as seguintes classes:

- $A = \{x \in \mathbb{N}^* : x = 1 + 26\} = \{1, 27, 53, 79, \cdots\}$
- $B = \{x \in \mathbb{N}^* : x = 2 + 26\} = \{2, 28, 54, 80, \cdots\}$
- $C = \{x \in \mathbb{N}^* : x = 3 + 26\} = \{3, 29, 55, 81, \cdots\}$
- $\vdots$
- $Z = \{x \in \mathbb{N}^* : x = 25 + 26\} = \{25, 51, 77, 103, \cdots\}$

Observe, pelos exemplos acima, que cada elemento do conjunto indica o marco inicial da contagem, isto é, cada elemento do conjunto indica a mesma hora em dias distintos, ou a mesma letra do alfabeto mas em ciclos distintos.

Será interessante nomearmos tais classes apenas por números ao invés de sua denotação intuitiva, pois nosso interesse será operar com essas classes. Dessa forma, vamos estipular que o nome da classe seja dado pela sua primeira marcação, por exemplo: em $A = \{x \in \mathbb{N}^* : x = 1 + 26\}$ o número 1 indica a primeira posição ocupada pela letra A, por isso a classe da letra A será representada pelo número $\overline{1}$. A barra é para diferenciar o nome da classe de seu elemento, o número 1.

A criptografia RSA usa essa mesma estratégia para iniciar a codificação de uma mensagem, por isso a aritmética modular será essencial para o todo o processo.

Primeiramente, precisamos deixar claro como nomear essas classes, estabelecer o tamanho do ciclo que está sendo adotado e o contexto em que desejamos trabalhar para então podermos definir as operações entre esses elementos.

Definição 1.2.1. Seja $\sim$ uma relação e X um conjunto, sendo x, y e z elementos de X. Uma classe em X é chamada de *equivalência* se para todo elemento de X as seguintes propriedades forem satisfeitas:

- Reflexividade: $x \sim x$;
- Simetria: se $x \sim y$, então $y \sim x$;
- Transitividade: se $x \sim y$ e $y \sim z$, então $x \sim z$.

Dizemos, nesse caso, que a relação $\sim$ é uma relação de equivalência.

Seja X um conjunto e $\sim$ uma relação de equivalência definida em X. Denotamos por $\overline{x}$ a classe de equivalência de x e escrevemos, em símbolos, da seguinte forma:

$$\overline{x} = \{y \in X : y \sim x\}$$

Como cada classe é composta por elementos distintos, para garantir a unicidade da soma precisamos que o seguinte princípio seja satisfeito:

Teorema 1.2.2. *Seja X um conjunto e $\sim$ uma relação de equivalência definida em X, então:*

$$\textit{Se } x \in X \textit{ e } y \in \overline{x} \textit{ então } \overline{x} = \overline{y}$$

Demonstração. Para mostrar a igualdade entre dois conjuntos devemos mostrar que $\overline{x} \subseteq \overline{y}$ e $\overline{y} \subseteq \overline{x}$ são verificadas. Vamos demonstrar o primeiro caso, o outro é análogo.

Tome $z \in \overline{x}$, então $z \sim x$. Dado que $y \in \overline{x}$ então $y \sim x$, daí pela propriedade simétrica temos que $x \sim y$. Logo, já que $z \sim x$ e $x \sim y$ então, pela transitividade, temos que $z \sim y$, isto é, $z \in \overline{y}$. Portanto, $\overline{x} \subseteq \overline{y}$. □

O resultado acima mostra que duas classes de equivalência não compartilham elementos, isto é, as classes de equivalência não têm elementos em comum, vide exemplos acima. Como consequência deste fato temos que o conjunto X é a união de todas as classes de equivalência.

O conjunto das classes de equivalência em X é chamado *conjunto quociente de X por $\sim$* e seus elementos são formados por subconjuntos de X separados pela relação de equivalência.

Nosso interesse será separar em classes de equivalência o conjunto dos números inteiros, dessa forma X representa o conjunto $\mathbb{Z}$, e x um número inteiro enquanto que $\sim$ representa alguma relação estabelecida entre os números inteiros.

Considere o exemplo em que as classes são compostas aqueles números que partilham do mesmo resto quando são divididos por 5. Neste caso podemos formar cinco classes distintas em $\mathbb{Z}$, a saber:

- Classe do resto zero: $\overline{0} = \{0, 5, 10, 15, 20, \cdots\}$
- Classe do resto um: $\overline{1} = \{1, 6, 11, 16, 21, \cdots\}$
- Classe do resto dois: $\overline{2} = \{2, 7, 12, 17, 22, \cdots\}$
- Classe do resto três: $\overline{3} = \{3, 8, 13, 18, 23, \cdots\}$
- Classe do resto quatro: $\overline{4} = \{4, 9, 14, 19, 24, \cdots\}$

No nosso exemplo $\{\overline{0}, \overline{1}, \overline{2}, \overline{3}, \overline{4}\}$ representa conjunto quociente de $\mathbb{Z}$ pela divisão por 5, o qual será denotado por $\mathbb{Z}_5$. Em termos gerais, o conjunto quociente de $\mathbb{Z}$ pela divisão por $n \neq 0$ é denotado por:

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

Pelo teorema de divisão (Teorema 1.1.1) sabemos que a divisão de a pode n é expressa como: $a = kn + r$, com $0 \leq r < k$. Dessa forma, podemos expressar esse fato como $a - r = kn$, para algum $k \in \mathbb{Z}$. Isso quer dizer que a diferença $a - r$ é um múltiplo de n. Será conveniente nos referirmos aos elementos de uma mesma classe em $\mathbb{Z}_n$ dessa maneira olhando para a diferença entre eles, como mostramos a seguir.

Dizemos que dois inteiros a e b são *congruentes módulo* n se a diferença $a - b$ é um múltiplo de n. Nesse caso, denotamos da seguinte forma:

$$a \equiv b \pmod{n}$$

Observe que $12 \equiv 22 \pmod{5}$, pois $12 - 22 = -10$ e -10 é um múltiplo de 5.

A definição acima formaliza a ideia de "pular de n em n" como é possível notar pelos exemplos dados. Vejamos mais alguns exemplos de elementos equivalentes segundo um determinado pulo:

- $8 \equiv 0 \pmod 4$, pois $8 - 0 = 8$ e 8 é um múltiplo de 4;
- $14 \equiv 24 \pmod 5$, pois $14 - 24 = -10$ e -10 é um múltiplo de 5;
- $25 \equiv 5 \pmod 5$, pois $25 - 5 = 20$ e 20 é um múltiplo de 5;

Observe que a congruência entre dois números depende do módulo esculhido, isto é, do tamanho do pulo que a ser dado. Pode-se mostrar que a conguência módulo n forma uma relação de equivalência. Não daremos essa demonstração aqui e o leitor interessado pode encontrá-la no capítulo 4 em [Cou14].

Nosso principal objetivo, agora, é mostrar como podemos operar com essas classes. Veremos que as operações em $\mathbb{Z}_n$ mantém as mesmas propriedade satisfeitas em $\mathbb{Z}$.

Definição 1.2.3. Sejam $\overline{a}, \overline{b} \in \mathbb{Z}_n$. As operações *adição*, *subtração* e *multiplicação* em $\mathbb{Z}_n$ são dadas por:

- **Adição:** $\overline{a} + \overline{b} = \overline{a + b}$;
- **Subtração:** $\overline{a} - \overline{b} = \overline{a - b}$;
- **Multiplicação:** $\overline{a} \times \overline{b} = \overline{a \times b}$;

Como estamos operando com classes, qualquer elemento da classe pode ser tomado como representando a classe a qual ele pertence, dessa forma, precisamos nos certificar de que o resultado dessas operações independem do representante da classe. Vejamos, por meio do exemplo, a validade desta propriedade para a soma.

Sejam $\overline{4}, \overline{3} \in \mathbb{Z}_5$. Queremos mostrar que a soma desses elementos independe do elemento tomado na classe como sendo seu representante.Tome 4 e 9 como representantes da classe $\overline{4}$ e tome 3 e 8 como representantes da classe $\overline{3}$. Então:

- $\overline{4} + \overline{3} = \overline{4 + 3} = \overline{7}$
- $\overline{4} + \overline{3} = \overline{9 + 8} = \overline{17}$

Como $7 \in \overline{2}$ e $17 \in \overline{2}$, então $\overline{7} = \overline{17} = \overline{2}$.

Sejam $\overline{a}, \overline{b}$ e $\overline{c}$ elementos de $\mathbb{Z}_n$. As seguintes propriedades são válidas para a adição e multiplicação:

$$
\begin{array}{rcll}
(\overline{a}+\overline{b})+\overline{c} & = & \overline{a}+(\overline{b}+\overline{c}) & \text{Associatividade da soma} \\
\overline{a}+\overline{b} & = & \overline{b}+\overline{a} & \text{Comutatividade da soma} \\
\overline{a}+\overline{0} & = & \overline{a} & \text{Elemento neutro da soma} \\
\overline{a}+\overline{-a} & = & \overline{0} & \text{Elemento oposto} \\
(\overline{a}\times\overline{b})\times\overline{c} & = & \overline{a}\times(\overline{b}\times\overline{c}) & \text{Associatividade do produto} \\
\overline{a}\times\overline{b} & = & \overline{b}\times\overline{a} & \text{Comutatividade do produto} \\
\overline{a}\times\overline{1} & = & \overline{a} & \text{Elemento neutro do produto} \\
\overline{a}\times(\overline{b}+\overline{c}) & = & \overline{a}\times\overline{b}+\overline{a}\times\overline{c} & \text{Distributividade}
\end{array}
$$

A demonstração dessas propriedades seguem imediatamente das propriedades correspondentes em $\mathbb{Z}$. Vejamos onde a aritmética modular se difere da aritimética em $\mathbb{Z}$. Considere as classes $\overline{2}$ e $\overline{6}$ em $\mathbb{Z}_6$. Claramente essas classes são diferentes da classe $\overline{0}$, no entanto vale que:

$$\overline{2}\times\overline{3}=\overline{6}=\overline{0}$$

Isso indica que a aritmética modular é diferente da aritmética em $\mathbb{Z}$. Fatos como esses permitem que certas propriedades que não são válidas em $\mathbb{Z}$ passem a valer em $\mathbb{Z}_n$, como veremos adiante.

Para completar o estudo das operações modulares está faltando definir a divisão entre os elementos de $\mathbb{Z}_n$. Mas, antes, lembre que em $\mathbb{Z}$ a frase "dividir a por b", para $b\neq 0$, é equivalente a dizer "multiplicar a por $\frac{1}{b}$", onde $\frac{1}{b}$ é chamado *inverso* de b. Em $\mathbb{Z}_n$ a divisão é definida nessa formulação equivalente, como mostramos a seguir.

Definição 1.2.4. Sejam $\overline{a},\overline{\alpha}\in\mathbb{Z}_n$. Dizemos que $\overline{\alpha}$ é o iverso de $\overline{a}$ se a seguinte equação for satisfeita em $\mathbb{Z}_n$:

$$\overline{a}\times\overline{\alpha}=1$$

A existência de elemento inverso em $\mathbb{Z}_n$ será de suma importância para a criptografia RSA, pois será justamente essa operação que irá permitir que uma mensagem seja decodificada uma vez que teremos um modo de fazer o percurso inverso. Para que isso seja possível será conveniente coletar todos os elementos que admitem inverso num mesmo conjunto para serem usados na codificação de uma mensagem, como veremos mais adiante.

Teorema 1.2.5 (Teorema de Inversão)**.** *A classe $\overline{a}$ tem inverso em $\mathbb{Z}_n$ se, e somente se, a e n não são primos entre si.*

Demonstração. ($\Longrightarrow$) Suponha que $\overline{a}$ tenha inverso em $\mathbb{Z}_n$, então existe um $\overline{\alpha}$ tal que $\overline{a} \times \overline{\alpha} = \overline{1}$. Isso é equivalente a dizer que $a \times \alpha - 1$ é divisível por n, ou seja, existe $k \in \mathbb{Z}$ tal que:

$$a \times \alpha + k \times n = 1 \tag{1.2}$$

Pelo Teorema 1.1.2 observamos que (1.2) afirma que $mdc(a, n) = 1$.

($\Longleftarrow$) Supondo que a seja um inteiro e que $mdc(a, n) = 1$. O Teorema 1.1.2 garante que existem α e β tais que $a \times \alpha + n \times \beta = 1$. Mas esta equação é equivalente a:

$$\overline{a} \times \overline{\alpha} = \overline{1}$$

□

Exemplo 1.2.6. Vejamos como encontrar o inverso de $\overline{3}$ em $\mathbb{Z}_{32}$.

Para saber se o inverso existe devemos aplicar o algoritmo euclidiano estendido para calcular o máximo divisor comum entre eles. Caso seja igual a 1, o algoritmo indicará qual é o inverso de $\overline{3}$ em $\mathbb{Z}_{32}$.

Restos	Quocientes	x	y
32	$*$	1	0
3	$*$	0	1
2	10	$x_1 = 1 - 10 \cdot 0 = 1$	$y_1 = 0 - 10 \cdot 1 = -10$
1	1	$x_2 = 0 - 1 \cdot 1 = -1$	$y_2 = 1 - 1(-10) = 11$
0	1	$*$	$*$

Portanto, $\alpha = -1$ e $\beta = 11$ e $mdc(32, 3) = (-1) \cdot (32) + (11) \cdot (3) = 1$.

Logo, o inverso existe. Como $3 \times 11 - 32 = 1$, isso equivale a dizer que:

$$\overline{3} \times \overline{11} = \overline{1} \text{ em } \mathbb{Z}_{32}$$

ou seja, $\overline{11}$ é o inverso de $\overline{3}$ em $\mathbb{Z}_{32}$. □

Como não são todos os elementos em $\mathbb{Z}_n$ que admitem inverso, será conveniente agrupá-los num mesmo conjunto para termos acesso a eles. Denotamos o conjunto dos elementos de $\mathbb{Z}_n$ que admitem inverso por $\mathcal{U}(n)$ e o descrevemos como:

$$\mathcal{U}(n) = \{\overline{a} \in \mathbb{Z}_n : mdc(a, n) = 1\}$$

É fácil calcular $\mathcal{U}(n)$ quando p é primo. De fato, como todo inteiro pode ser escrito de maneira única como produto de primos, sendo p um fator de a temos que p divide a, isso significa que $a \in \overline{0}$ em $\mathbb{Z}_p$. Caso contrário p não divide a, nesse caso como p é primo então $mdc(a, p) = 1$, portanto $\overline{a}$ admite inverso em $\mathbb{Z}_p$. Dessa forma, para p primo todas as classes distintas da classe $\overline{0}$ admitem inverso, isto é:

$$\mathcal{U}(p) = \mathbb{Z}_p - \overline{0}, \text{para } p \text{ primo}$$

Uma propriedade importante de $\mathcal{U}(n)$ é que ele é fechado por produto, isto é, o produto de dois elementos de $\mathcal{U}(n)$ está em $\mathcal{U}(n)$ e o inverso do produto é dado pelo produto dos inversos, isto é, se $\overline{\alpha}$ e $\overline{\beta}$ são respectivamente os inversos de $\overline{a}$ e $\overline{b}$, então $\overline{\alpha} \times \overline{\beta}$ é o inverso de $\overline{a} \times \overline{b}$.

É possível determinar a quantidade de elementos do conjunto $\mathcal{U}(n)$ usando a chamada *função totiente* ou *função de Euler* $\phi(n)$, a qual denota a ordem de $\mathcal{U}(n)$. Para nosso propósito vamos precisar apenas tratar do conjunto $\mathcal{U}(p)$ para p um número primo e isso nos polpará de uma série de resultados técnicos envolvendo teoria de grupos. Para detalhes acerca deste tema sugerimos o capítulo 8 de [Cou14].

Como vimos acima, $\mathcal{U}(p)$ tem $p-1$ elementos pois a classe $\overline{0}$ não está em $\mathcal{U}(n)$, daí para p primo temos que a função totiente resulta em:

$$\phi(p) = (p - 1)$$

Não é difícil calcular $\phi(p^k)$, pois sendo p um número primo, temos que a quantidade de inteiros positivos menores que p^k e que não o dividem, isto é, cujo máximo divisor comum é 1, é dado por:

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

O seguinte teorema, o qual não demonstraremos aqui, mostra como generalizar os resultados acima mencionados.

Teorema 1.2.7. *Se m, n são inteiros positivos tais que $mdc(m, n) = 1$, então*

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

Demonstração. Confira a seção 4, capítulo 8 em [Cou14]. □

Na criptografia RSA usaremos o resultado acima para construir a chave de codificação de uma mensagem como mostraremos no próximo capítulo.

Nesse momento temos a nossa disposição todos os resultados que precisamos para construir o algoritmo RSA, mas vamos usar o chamado *teorema de Fermat* para efetuar simplicações de números envolvendo potências muito altas. O teorema de Fermat aparece em duas versões e para o nosso propósito vamos precisar apenas da versão chamada *pequeno teorema de Fermat.*

Teorema 1.2.8 (Teorema de Fermat)**.** *Seja p um número primo e a um número inteiro, então*

$$a^p \equiv a \pmod{p}$$

Demonstração. Confira na seção 3 do capítulo 5 em [Cou14]. □

O teorema afirma que se p é primo e a um inteiro qualquer, então p divide a diferença $a^p - a$ em $\mathbb{Z}_p$. A outra versão sai como consequência do teorema anterior como mostramos a seguir.

Teorema 1.2.9 (Pequeno Teorema de Fermat)**.** *Seja p um número primo e a um número inteiro que não é divisível por p. Então:*

$$a^{p-1} \equiv 1 \pmod{p}$$

Demonstração. Se p não divide a, então pelo Teorema 1.2.5 sabemos que a admite inverso em $\mathcal{U}(p)$, daí existe $\overline{\alpha} \in \mathcal{U}(p)$ tal que $\overline{a} \cdot \overline{\alpha} = \overline{1}$, isto é, $a \cdot \alpha \equiv 1 \pmod{p}$. Multiplicando por α ambos os membros de $a^p \equiv a \pmod{p}$ dada no Teorema 1.2.8 obtemos $\alpha \cdot a^p \equiv \alpha \cdot a \pmod{p}$, o que equivale a:

$$\alpha \cdot a \cdot a^{p-1} \equiv \alpha \cdot a \pmod{p} \tag{1.3}$$

Substituindo $a \cdot \alpha \equiv 1 \pmod{p}$ em (1.3) obtemos que $a^{p-1} \equiv 1 \pmod{p}$. □

O Teorema de Fermat é usado para simplificar os cálculos na obtenção de congruências e isso será muito útil porque teremos que trabalhar com números muito grandes para poder garantir a segurança na criptografia RSA.

Vejamos como aplicar o teorema para simplificar os cálculos na prática.

Para calcular $2^{5432675}$ módulo 13, usando o Teorema 1.2.9 basta calcular o resto da divisão de 5432675 por $12 = 13 - 1$, que resulta em 11. Assim, temos que $2^{5432675} \equiv 2^{11} \equiv 7 \pmod{13}$.

Outro resultado usado para decodificar uma mensagem na criptografia RSA é o chamado *Teorema Chinês do resto*. Esse teorema é útil para resolver sistemas de congruências, isto é, sistemas envolvendo situações com ciclos, por exemplo: para saber quando três satélites que giram em torno da Terra com ciclos diferentes estarão alinhados usamos o teorema chinês do resto.

Na prática, o teorema chinês do resto é um procedimento usado para encontrar uma classe de equivalência adequada para ser solução do sistema, mas em uma outra base. Uma equação envolvendo congruências é dada pela expressão:

$$ax \equiv b \pmod{n} \tag{1.4}$$

Queremos determinar o valor de x de modo a tornar verdadeira a equação em $\mathbb{Z}_n$. Primeiramente temos que considerar duas possibilidades: (i) $\overline{a}$ tem inverso em $\mathbb{Z}_n$; (ii) $\overline{a}$ não tem inverso em $\mathbb{Z}_n$.

(i) Se $\overline{a}$ tem inverso, então existe $\overline{\alpha} \in \mathbb{Z}_n$ tal que $\overline{a} \cdot \overline{\alpha} \equiv \overline{1}$. Daí, se multiplicarmos a equação (1.4) por $\overline{\alpha}$ resulta que $x \equiv \alpha \cdot b \pmod{n}$ é a solução da equação.

(ii) Se $\overline{a}$ não tem inverso em $\mathbb{Z}_n$, então $mdc(a, n) = d \neq 1$. Dizer que a equação (1.4) tem solução significa que existem inteiros x e y tais que

$$ax - ny = b \tag{1.5}$$

Por um lado, como d divide a e d divide n, então d divide b. Daí, existem inteiros a', b' e c' tais que $a = da'$, $b = db'$ e $n = dn'$. Substituindo essas identidades em (1.5) segue que $da'x - dn'y = db'$. Dividindo toda a expressão por d temos que $a'x - n'y = b'$ e isso significa que

$$a'x \equiv b' \pmod{n'} \tag{1.6}$$

Agora, como $mdc(a, n) = d$ então pelo Teorema 1.1.2 temos que existem inteiros α e β tais que $a\alpha + n\beta = d$, daí substituindo as identidades obtidas acima temos que $da'\alpha + dn'\beta = d$, daí dividindo toda a expressão por d segue que $a'\alpha + n'\beta = 1$, isto é $mdc(a', n') = 1$, logo $\overline{a'}$ admite inverso em $\mathbb{Z}_{n'}$, o qual chamamos de $\overline{\gamma}$. Portanto, multiplicando a expressão (1.6) por $\overline{\gamma}$ segue que $x \equiv \gamma \cdot b' \pmod{n'}$ é a solução da equação.

Para decodificar uma mensagem criptografada pelo RSA será necessário resolver sistemas envolvendo apenas duas equações de congruências por isso

optamos por não trabalhar com o caso geral de sistemas envolvendo mais de duas equações nesta monografia.

Considere o seguinte sistema:

$$\begin{aligned} x &\equiv a \pmod{m} \\ x &\equiv b \pmod{n} \end{aligned}$$

O sistema pode ser resolvido seguindo os seguintes passos:

1. A primeira equação pode ser reescrita na forma $x = a+my$, para algum inteiro y;
2. Substituindo a expressão obtida no item anterior na segunda equação temos: $a + my \equiv b \pmod{n}$;
3. A equação anterior pode ser reescrita na forma $my \equiv (b-a) \pmod{n}$;
4. Resolva a última equação usando o procedimento descrito anteriormente e encontre o valor de y.
5. Para encontrar a solução basta substituir o valor de y na equação dada no passo 1.

Considere, como exemplo, o problema de calcular o menor inteiro x cujo resto é 1 na divisão por 3, e cujo resto é 2 na divisão por 5. Esse problema pode ser descrito pelo seguinte sistema de equações:

$$\begin{aligned} x &= 1 \pmod{3} \\ x &= 2 \pmod{5} \end{aligned}$$

A primeira equação pode ser escrita como $x = 1+3y$ para algum inteiro y. Substitua o valor de x na segunda equação e obtenha $1 + 3y = 2 \pmod{5}$, isto é $3y = 1 \pmod{5}$.

Como $mdc(3, 5) = 1$ então multiplicando toda a expressão pelo inverso de $\overline{3}$ em $\mathbb{Z}_5$, a saber, por $\overline{2}$, obtemos que $y = 2 \pmod{5}$. Isso significa que $y = 2 + 5k$ para $k \in \mathbb{Z}$. Daí, substituindo em $x = 1 + 3y$ obtemos que $x = 1 + 3(2 + 5k)$, isto é, $x = 7 + 15k$, ou seja, $x \equiv 7 \pmod{15}$.

O procedimento exemplicado acima pode ser sintetizado pelo seguinte teorema:

Teorema 1.2.10 (Teorema Chinês do Resto)**.** *Sejam $n_1, n_2, \cdots n_k$ inteiros positivos dois a dois primos entre si. O sistema*

$$\begin{aligned} x &\equiv a_1 \pmod{n)_1} \\ x &\equiv a_2 \pmod{n)_2} \\ &\vdots \\ x &\equiv a_k \pmod{n)_k} \end{aligned}$$

sempre tem uma solução em $\mathbb{Z}_{n_1 \cdots n_k}$.

Demonstração. Veja a seção 5 do capítulo 7 em [Cou14]. □

Para encerrarmos este capítulo vejamos como podemos combinar o teorema de Fermat (Teorema 1.2.8) e o teorema chinês do resto (Teorema 1.2.10) a fim de simplificar o cálculo de potências módulo n. Mostraremos esse procedimento por meio de um exemplo.

O primeiro passo será fatorar o número que aparece no módulo. Vamos supor que todos os fatores primos que ocorrem na fatoração tem multiplicidade igual a 1, pois esse será o caso que vamos utilizar neste trabalho. Claramente os fatores primos são dois a dois primos entre si, daí o Teorema 1.2.10 afirma que existe uma única solução para um sistema cujos os módulos são dados por tais fatores. Para cada fator primo obtido na fatoração iremos encontrar a classe a qual a potência dada no problema pertence usando o Teorema 1.2.8, e com isso podemos construir um sistema cuja solução resultará na simplificação da potência dada.

Vejamos como calcular $2^{6754} \pmod{1155}$.

Fatorando 1155 temos que $1155 = 3 \times 5 \times 7 \times 11$. Agora, vamos aplicar o Teorema 1.2.9 a cada um dos primos, obtendo as seguintes equivalências:

$$\begin{aligned} 2^2 &\equiv 1 \pmod{3} \\ 2^4 &\equiv 1 \pmod{5} \\ 2^6 &\equiv 1 \pmod{7} \\ 2^{10} &\equiv 1 \pmod{11} \end{aligned}$$

Agora, dividindo 6754 por $p-1$, para $p \in \{3, 5, 7, 11\}$, temos:

$$\begin{aligned} 6754 &= 2 \times 3377 + 0 \\ 6754 &= 4 \times 1688 + 2 \\ 6754 &= 6 \times 1125 + 4 \\ 6754 &= 10 \times 675 + 4 \end{aligned}$$

Em seguida substituímos as identidades acima nas congruências e as reduzimos

$$\begin{array}{rcrcll} 2^{6754} & = & 2^{2\times 3377} & = & (2^{3377})^2 & \equiv 1 \pmod 3 \\ 2^{6754} & = & 2^{4\times 1688+2} & = & (2^{1688})^4 & \equiv 4 \pmod 5 \\ 2^{6754} & = & 2^{6\times 1125+4} & = & (2^{1125})^6 & \equiv 2 \pmod 7 \\ 2^{6754} & = & 2^{10\times 675+4} & = & (2^{675})^{10} & \equiv 5 \pmod{11} \end{array}$$

Logo, a tarefa, agora, consiste em resolver o seguinte sistema:

$$\begin{array}{rcll} x & \equiv & 1 & \pmod 3 \\ x & \equiv & 4 & \pmod 5 \\ x & \equiv & 2 & \pmod 7 \\ x & \equiv & 5 & \pmod{11} \end{array}$$

Agora, basta resolver o sistema usando o algoritmo chinês.

1. A primeira congruência equivale a dizer que $x = 3y + 1$;

2. Substituindo x na segunda congruência temos que $3y+1 \equiv 4 \pmod 5$. Isso equivale a dizer que $y \equiv 1 \pmod 5$, isto é, $y = 5k+1$, para algum $k \in \mathbb{Z}$;

3. Substituindo o valor de y em $x = 3y + 1$ temos $x = 4 + 15z$.

4. Substitua o novo valor de x na terceira congruência e obtenha $4 + 15z \equiv 2 \pmod 7$. Isso equivale a $15z \equiv -2 \pmod 7$, isto é, $z \equiv 5 \pmod 7$ e daí $z = 7t + 5$, para $t \in \mathbb{Z}$.

5. Agora, substituindo o valor de z em $x = 4 + 15z$ obtemos que $x = 4 + 15(7t + 5)$, isto é, $x = 105t + 79$.

6. Substituindo o valor de x que acabamos de obter na última congruência temos $105t + 79 \equiv 5 \pmod{11}$, isto é, $t \equiv 6 \pmod{11}$, o que resulta em $x = 709 + 1155u$ para $u \in \mathbb{Z}$. Concluímos com isso que $2^{6754} \equiv 709 \pmod{1155}$.

Agora estamos em condições de apresentar o funcionamento do algorítmo RSA, o que faremos no próximo capítulo.

2. APLICANDO A CRIPTOGRAFIA RSA

A criptografia RSA tem suma importância para toda a comunicação moderna. Ela é tão importante que a descoberta de uma forma de se desencriptá-la colocaria em risco a sociedade como a conhecemos. Ao longo deste capítulo vamos ver como é seu funcionamento usando os conteúdos do capítulo anterior.

2.1 Preparando-se para criptografar

Para que o algoritmo RSA possa encriptar de forma eficiente, precisaremos seguir uma série de passos necessários para que o RSA funcione, mas que ainda não são parte do algoritmo.

O primeiro passo é a conversão das letras da mensagem em números. A essa etapa chamaremos de pré-codificação. Para que o RSA venha a funcionar, precisamos seguir uma tabela como a apresentada abaixo:

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	X	Y	W	Z
23	24	25	26	27	28	29	30	31	32	33	34	35

Para representar espaços vamos usar o 99. Avisamos que esta é uma tabela apenas com finalidade didática, e, por isso há vários caracteres desconsiderados. Como exemplo vamos pré-encriptar o poema Amor, de Oswald de Andrade. O texto do poema a ser pré-encriptado é o seguinte:

Amor
Humor.

Como primeiro passo vamos converter todas as letras em números, resultando em:

10 22 24 27 99 17 30 22 24 27

Feito isso, agrupamos o conjunto em um bloco único de caracteres:

10222427991730222427

Atente-se ao fato de todo o caractere convertido possuir sempre o mesmo número de algarismos. Isso é útil para evitar ambiguidades na fase de desencriptação.

Nosso próximo passo nesta fase que antecede a encriptação consiste em definir quais serão os primos p e q. Para nosso exemplo vamos usar $p = 17$ e $q = 23$, como mencionado na Introdução deste artigo, temos que $n = pq$, logo $n = 391$.

O último passo da pré-encriptação consiste em quebrar o número que obtemos acima em blocos menores. Esses blocos devem obedecer à duas regras básicas: serem menores que n, ou no nosso exemplo 391, pois iremos trabalhar com o módulo 391 durante a encriptação, e não podem se iniciar por 0, para não haver ambiguidades na desencriptação. Vejamos como a nossa mensagem fica quando pré-encriptada.

102 — 224 — 279 — 91 — 7 — 30 — 222 — 42 — 7

Perceba que não há relação entre nenhum dos números obtidos com um caractere específico, o que torna impossível a associação de um número a uma letra por frequência de aparecimento.

2.2 Codificando e decodificando mensagens

Encerrada a fase de pré-codificação vamos agora codificar nossas mensagens. Manteremos os valores e exemplos da seção anterior a fim de facilitar a compreensão.

2.2.1 Codificando uma mensagem

A esta altura nós já conhecemos o número n, que em nosso exemplo possui o valor de 391. O outro número que iremos usar será o $\mathbf{e}$. Tomaremos que o $mdc(\mathbf{e}, \phi(n)) = 1$. Para calcularmos o valor de $\phi(n)$, conforme já comentamos em 1.2.7 precisaremos aplicar a seguinte receita:

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

Que em nosso exemplo resulta em:

$$\phi(391) = (17-1)(23-1) = 16 \cdot 22 = 352$$

Para determinarmos o $\mathbf{e}$ basta escolher o menor primo que $mdc(\mathbf{e}, 352) = 1$, que no nosso caso será o 3. Optamos por um primo não múltiplo ao invés de um composto para podermos aplicar o teorema de Fermat (Teorema 1.2.9) mais adiante. Ao conjunto $(n, \mathbf{e})$ denominamos chave de encriptação.

Vamos chamar o bloco codificado que iremos encriptar de b, lembrando que b é um número inteiro menor que n. Tambem chamaremos o bloco após a codificação de $C(b)$. Para obtermos $C(b)$ devemos aplicar a seguinte fórmula:

$$C(b) \equiv b^{\mathbf{e}} \pmod{n}$$

Podemos para facilitar dizer que $C(b)$ é o resíduo de $b^{\mathbf{e}}$ pelo módulo n. Vamos à uma demonstração prática com o primeiro bloco de nossa mensagem, que possui o valor 102. Para simplificar o nosso trabalho vamos utilizar as operações modulares.

$$102^3 \equiv 24276 \equiv 34 \pmod{391}$$

Faremos o mesmo procedimento para todos nossos blocos:

$$\begin{array}{rcrcrl}
224^3 & \equiv & 11239424 & \equiv & 129 & \pmod{391} \\
279^3 & \equiv & 21717639 & \equiv & 326 & \pmod{391} \\
91^3 & \equiv & 753571 & \equiv & 114 & \pmod{391} \\
7^3 & \equiv & 343 & \equiv & 343 & \pmod{391} \\
30^3 & \equiv & 27000 & \equiv & 21 & \pmod{391} \\
222^3 & \equiv & 10941048 & \equiv & 86 & \pmod{391} \\
42^3 & \equiv & 74088 & \equiv & 189 & \pmod{391} \\
7^3 & \equiv & 343 & \equiv & 343 & \pmod{391}
\end{array}$$

Portanto, “Amor Humor”, encriptado pelo RSA com as chaves $(391, 3)$ é:

$$34 \text{ — } 129 \text{ — } 326 \text{ — } 114 \text{ — } 343 \text{ — } 21 \text{ — } 86 \text{ — } 189 \text{ — } 343$$

2.2.2 Decodificando uma mensagem

Para podermos desencriptar uma mensagem nós precisamos de dois números. O primeiro é a nossa chave pública n. O segundo número é d, que consiste no inverso de $\mathbf{e}$ no módulo $\phi(n)$, ou seja $\mathbf{e} \cdot d \equiv 1 \pmod{\phi(n)}$ (vide Teorema 1.2.5). Para o nosso exemplo $d = 235$.

Agora que já conhecemos a d podemos decodificar a mensagem. Vamos fazer isso em nosso primeiro bloco codificado, que possui o valor 34. Para achar a resposta precisaremos calcular $D(34) \equiv 34^{235} \pmod{391}$. Esse cálculo seria praticamente impossível sem o uso dos Teoremas: chinês do resto (Teorema 1.2.10) e de Fermat (Teorema 1.2.9).

Nosso primeiro passo será o de calcular 34^{235} nos módulos 17 e 23, que são os primos resultantes da fatoração de n. Neste caso, começamos com:

$$34 \equiv 0 \pmod{17}$$

$$34 \equiv 11 \pmod{23}$$

Assim teremos que $34^{235} \equiv 0^{235} \equiv 0 \pmod{17}$. Aplicando o teorema de Fermat (Teorema 1.2.9) na outra congrência teremos:

$$11^{235} \equiv (11^{22})^{10} \cdot 11^{15} \equiv 11^{15} \pmod{23}$$

Como $11 \equiv -12 \equiv -4 \cdot 3 \pmod{23}$, nós podemos afirmar que:

$$11^{235} \equiv 11^{15} \equiv -4^{15} \cdot 3^{15} \pmod{23}$$

Com isso, teremos:

$$4^{15} \equiv 2^{30} \equiv (2^{11})^2 \cdot 2^8 \equiv 2^8 \equiv 3 \pmod{23}$$

$$3^{15} \equiv 3^{11} \cdot 3^4 \equiv 3^4 \equiv 12 \pmod{23}$$

Concluindo assim:

$$112^{35} \equiv -4^{15} \cdot 3^{15} \equiv -3 \cdot 12 \equiv 10 \pmod{23}$$

Temos assim as congruências $34^{235} \equiv 0 \pmod{17}$ e $34^{235} \equiv 10 \pmod{23}$. Com isso podemos aplicar o teorema chinês do resto (Teorema 1.2.10) no sistema:

$$x \equiv 0 \pmod{17}$$

$$x \equiv 10 \pmod{23})$$

Com ele iremos obter:

$$10 + 23y \equiv 0 \pmod{17})$$

Obtendo assim:

$$6y \equiv 7 \pmod{17}$$

Porém, 3 é o inverso de 6 no módulo 17, e por isso teremos:

$$y \equiv 3 \cdot 7 \equiv 4 \pmod{17}$$

Com isso iremos chegar até $x = 10 + 23y = 10 + 23 \cdot 4 = 102$. Caso você venha a conferir na seção sobre codificação de mensagens poderá confirmar o resultado. Os demais blocos podem ser decodificados da mesma forma, apenas não serão mostrados neste artigo por necessitar de muitos passos, o que tornaria este capítulo inutilmente mais longo.

2.3 Provando a funcionalidade do RSA

Ao longo desta seção vamos provar que o RSA funciona no processo de decodificação. Para podermos fazer isso teremos que provar que:

$$b \equiv D(C(b)) \pmod{n}$$

Nós já vimos que $C(b) \equiv b^{\mathbf{e}} \pmod{n}$ e $D(b) \equiv b^d \pmod{n}$. Se combinarmos ambas as congruência iremos obter:

$$D(C(b)) \equiv (b^{\mathbf{e}})^d = b^{ed} \pmod{n}$$

Logo o que temos de provar é que $b^{\mathbf{e}d} \equiv b \pmod{n}$. Como por definição $\mathbf{e}d \equiv 1 \pmod{(p-1)(q-1)}$, nós veremos que:

$$\mathbf{e}d = 1 + k(p-1)(q-1)$$

Pelo Teorema Chinês do Resto (Teorema 1.2.10) e levando em conta a expressão para obter $\mathbf{e}d$ temos que:

$$b^{\mathbf{e}d} \equiv b(b^{p-1})^{k(q-1)}$$

Tomando que p não divide b, nós podemos vir usar o Teorema de Fermat (Teorema 1.2.9), de modo a obter:

$$b^{p-1} \equiv 1 \pmod{p}$$

Obtendo assim:

$$b^{\mathbf{e}d} \equiv b \cdot (1)^{k(q-1)} \equiv b \pmod{p}$$

Mesmo considerando que b seja múltiplo de p, teremos que b e b^{ed} são congruentes a 0, logo nesse caso também é válida a congruência, tendo assim:

$$b^{\mathbf{e}d} \equiv b \pmod{p}$$

Pelo mesmo método podemos obter q, obtendo o par:

$$b^{\mathbf{e}d} \equiv b \pmod{p}$$
$$b^{\mathbf{e}d} \equiv b \pmod{q}$$

Veja que b é solução de:

$$x \equiv b \pmod{p}$$
$$x \equiv b \pmod{q}$$

Pelo Teorema Chinês do resto (Teorema 1.2.10) esse sistema tem solução igual:

$$b + p \cdot q \cdot t$$

Onde $t \in \mathbb{Z}$. Logo, como provamos anteriormente, temos que:

$$b^{\mathbf{e}d} \equiv b + p \cdot q \cdot k$$

Para algum inteiro k. Caso tomemos $k = 0$ teremos $b^{\mathbf{e}d} \equiv b \pmod{p}$, Comprovando $b = D(C(b))$.

2.4 Discutindo a segurança do RSA

Antes de mudarmos o foco deste trabalho, vamos prestar atenção no que tange a segurança do RSA. Vamos supor que alguém, que vamos denominar Irineu, esteja com uma escuta em nossas mensagens, tendo assim acesso tanto à mensagem codificada quanto à chave pública n. Vamos lembrar que n é a multiplicação dos primos p e q. Sabendo disso, bastaria para Irineu fatorar n para obter p e q e depois descobrir d para poder decodificar a mensagem, como já foi explicado neste capítulo.

Isso pode parecer muito simples, mas como já mostramos na seção sobre fatoração, não há um algoritmo conhecido que possa fazer isso de forma eficiente. O que ocorre é que um algoritmo que faça a fatoração de forma eficiente pode vir a surgir a qualquer momento, do ponto que não há nenhuma prova matemática de que esse algoritmo não exista.

O que iremos fazer nos próximo capítulo, consiste em analisar a viabilidade de uma variação da criptografia RSA que não seja completamente vulnerável caso uma fatoração eficiente no conjunto $\mathbb{Z}$ seja descoberta: a *Criptografia RSA Gaussiana*.

3. INTEIROS E PRIMOS DE GAUSS

Até o momento apenas os números inteiros foram abordados neste projeto, mas para podermos entender a RSA Gaussiana é necessário conhecer o conjunto dos números inteiros gaussianos. Ao longo deste capítulo vamos conhecer os inteiros e os primos gaussianos e suas propriedades aritméticas básicas.

3.1 Inteiros de Gauss e suas propiedades

Os inteiros gaussianos, conjunto que a partir de agora iremos nos referenciar por $\mathbb{Z}[i]$, são um subconjunto dos números complexos, relembrando que os números complexos são os números de forma $a + b\mathbf{i}$, onde a e b são reais e $\mathbf{i}$ é a $\sqrt{-1}$. A diferença entre o conjunto $\mathbb{Z}[i]$ e o conjunto C reside no fato de em $\mathbb{Z}[i]$ a e b serem números inteiros. Formalmente dizemos que os inteiros gaussianos são:

$$\mathbb{Z}[i] = \{a + b\mathbf{i} | a, b \in \mathbb{Z}\}, \text{ onde } \mathbf{i}^2 = -1$$

Por $\mathbb{Z}[i]$ estar contido em $\mathbb{C}$, as operações deste conjunto podem ser realizadas, por exemplo, se tomarmos $z_1 = a + b\mathbf{i}$ e $z_2 = c + d\mathbf{i}$ nós iremos obter:

$$z_1 + z_2 = (a + c) + (b + d)\mathbf{i}$$
$$z_1 \cdot z_2 = (ac - bd) + (ad + bc)\mathbf{i}$$

Outra propiedade herdada é a dos elementos neutros, o $0 = 0+0\mathbf{i}$ continua sendo o elemento neutro da adição, enquanto o $1 = 1 + 0\mathbf{i}$ também continua sendo o elemento neutro da multiplicação. As propiedades associativa da adição e da multiplicação, comutativa da adição e multiplicação e distributiva também são herdadas do conjunto complexo.

Repare que se considermos o plano complexo, os inteiros gaussianos terão uma marcação reticulada. Outro conceito importante para os inteiros gaussianos é a norma do número, ela é importante para auxiliar na definição de um primo gaussiano, assim como são importantes os conceitos de número conjugado e número associado. Caso venhamos a tomar um número inteiro gaussiano de forma $a + b\mathbf{i}$, sua norma será $a^2 + b^2$.

Definição 3.1.1. A norma de um número gaussiano é a soma dos quadrados de seus valores absolutos como número complexo. Ela é o resultado de:

$$N(a + b\mathbf{i}) = a^2 + b^2 = (a + b\mathbf{i})(a - b\mathbf{i}),$$

onde o $(a - b\mathbf{i})$ é a conjugado de $(a + b\mathbf{i})$, também denotado por $\overline{(a + b\mathbf{i})}$.

Uma das propiedades da norma é ser multiplicativa, ou seja, a norma de $N(zw)$ é igual a $N(z) \cdot N(w)$.

Os inteiros gaussianos possuem como unidades básicas ± 1 e $\pm\mathbf{i}$. Caso venhamos a multiplicar um inteiro gaussiano x, teremos que $\pm x$ e $\pm x\mathbf{i}$ sendo seus elementos associados.

Definição 3.1.2. Os elementos associados de um número x, tal que $x \in \mathbb{Z}[i]$, são $\pm x$ e $\pm x\mathbf{i}$.

Para chegarmos aos primos Gaussianos precisaremos demonstrar para o conjunto $\mathbb{Z}[i]$ uma série de resultados que já são conhecidos do conjunto dos números inteiros, como o funcionamento da divisão e o teorema da fatoração única.

Podemos definir a divisibilidade gaussiana por quando dizemos que β divide α, representado por $\beta|\alpha$ se $\alpha = \beta\gamma$, para qualquer $\gamma \in \mathbb{Z}[i]$. Nesse caso, β é um fator de α.

Teorema 3.1.3. *Um inteiro Gaussiano $\alpha = a + b\boldsymbol{i}$ é dividido por um primo c se e somente se $c|a$ e $c|b$ em $\mathbb{Z}$.*

Demonstração. Dizer que $c|(a + b\mathbf{i})$ em $\mathbb{Z}$ é o mesmo que dizer que $a + b\mathbf{i} = c(m + n\mathbf{i})$, para algum $m, n \in \mathbb{Z}$, que equivale a $a = cm$ e $b = cn$.

□

Tomemos uma divisão entre inteiros gaussianos, onde α é o dividendo, β o divisor, γ o quociente e ρ o dividendo

Teorema 3.1.4 (Teorema da divisão conjunto gaussiano)**.** *Para $\alpha, \beta \in \mathbb{Z}$ com $\beta \neq 0$ existe um $\gamma, \rho \in \mathbb{Z}$ tal qual $\alpha = \beta\gamma + \rho$ e $N(\rho) < N(\beta)$. De fato, podemos escolher ρ de forma que $N(\rho) \leq (1/2)N(\beta)$*

Agora que já entendemos a divisão, vamos definir o máximo divisor comum no conjunto $\mathbb{Z}[i]$.

Teorema 3.1.5 (Algoritmo Euclidiano no conjunto gaussiano)**.** *Tomemos $\alpha, \beta \in \mathbb{Z}[i]$ e diferentes de 0. Aplicamos recursivamente o teorema da divisão em $\mathbb{Z}[i]$ (3.1.4), começando com esse par e fazendo com o resto uma equação com um novo dividendo e divisor no próximo caso, enquanto o resto for diferente de zero:*

$$\begin{array}{rcll} \alpha & = & \beta\gamma_1 + \rho_1, & N(\rho_1) < N(\beta) \\ \beta & = & \rho_1\gamma_2 + \rho_2, & N(\rho_2) < N(\rho_1) \\ \rho_1 & = & \rho_2\gamma_3 + \rho_3, & N(\rho_3) < N(\rho_2) \\ & \vdots & & \end{array}$$

O último elemento que não possua resto 0 é divisível por todos os divisores comuns de α e β, sendo esse o maior divisor comum de α e β.

Outra possibilidade para simplificar esta mesma operação é o uso do algoritmo eucliadiano extendido, para o conjunto dos $\mathbb{Z}[i]$ esse algoritmo é chamado de Teorema de Bezout.

Corolário 3.1.6. *Para α e β diferentes de 0 e existentes no conjunto gaussiano, tomemos δ como o maior divisor comum pelo algoritmo euclidiano no conjunto gaussiano(3.1.5). Qualquer divisor comum de α e β é um múltiplo de δ.*

Demonstração. Tomemos δ' como o maior divisor de α e β. Pelo algoritmo euclidiano no conjunto gaussiano(3.1.5), $\delta' | \delta$ (pois δ' é divisor comum). Tendo que $\delta = \delta'\gamma$, então:

$$N(\delta) = N(\delta')N(\gamma) \geq N(\delta')$$

Tendo δ' como o maior divisor comum, sua norma é a maior entre os divisores comuns, logo a inequação $N(\delta) \geq N(\delta')$ tem de ser uma igualdade. Isso implica que $N(\gamma) = 1$, então $\gamma = \pm 1$ ou $\pm\mathbf{i}$. Então δ e δ' são múltiplos um do outro.

□

Teorema 3.1.7. *Sendo δ o maior divisor comum de dois inteiros gaussianos diferentes de zero α e β, então $\delta = \alpha x + \beta y$ para qualquer $x, y \in \mathbb{Z}$.*

Demonstração. Sendo δ escrito com uma combinação em $\mathbb{Z}[i]$ de α e β, ele não é afetado por substituir δ como o múltiplo por uma unidade. Por isso o Corolário 3.1.6, nós apenas temos que provar que δ é o maior divisor comum pelo algoritmo euclidiano no conjunto gaussiano Para δ, uma substituição no algoritmo euclidiano mostra que δ é uma combinação em $\mathbb{Z}[i]$ de α e β. Todos os demais detalhes são idêticos aos da prova para inteiros.

□

Podemos dizer que se α e β possuem apenas as unidades como fatores em comum eles são primos entre si.

Corolário 3.1.8. *Os inteiros gaussianos α e β são primos entre si se e somente se podemos escrever:*

$$1 = \alpha x + \beta y$$

para quaisquer $x, y \in \mathbb{Z}[i]$

Demonstração. Se α e β são primos entre si, então 1 é o maior divisor de α e β, então $1 = \alpha x + \beta y$ para qualquer $y \in \mathbb{Z}[i]$ pelo teorema 3.1.7, por outro lado se $1 = \alpha x + \beta y$ para algum $x, y \in \mathbb{Z}[i]$, então o máximo divisor comum de α e β é divisor de 1, logo uma unidade, provando que α e β são primos entre si.

□

Agora nós vamos definir o que vem a ser um inteiro gaussiano primo e composto, além de falarmos sobre a fatoração única.

Definição 3.1.9. Se tomarmos um inteiro Gaussiano α com $N(\alpha) > 1$. é denominado *composto* se o número possuir um fator não trivial. Caso ele possua apenas fatores triviais ele é denominado *primo*.

Agora que nós já sabemos o que é um número primo e composto, podemos definir como eles são fatorados de forma única pelo teoremas abaixo, as provas de ambas podem ser lidas na sessão 6 de "The Gaussian Integers" em [Con08], a partir da página 13. As provas dos outros teoremas desta sessão também foram baseadas no trabalho de [Con08].

Teorema 3.1.10. *Todo $\alpha \in \mathbb{Z}[i]$ com $N(\alpha) > 1$ é um produto de primos em $\mathbb{Z}[i]$*

Teorema 3.1.11 (Fatoração Única no conjunto gaussiano)**.** *Todo $\alpha \in \mathbb{Z}[i]$ com $N(\alpha) > 1$ possui uma única fatoração baseada nos primos gaussianos com o formato:*

$$\alpha = \pi_1\pi_2 \cdots \pi_r = \pi'_1\pi'_2 \cdots \pi'_{s'}$$

onde os π_i e os π'_j são primos em $\mathbb{Z}[i]$, então $r = s$ e cada membro de π_i após uma renumeração adequada é um múltiplo por uma unidade de π'_i.

Para exemplificar o que foi dito pelo Teorema da Fatoração Única (3.1.11), tomemos o número 2. Esse número é fatorado como $(1 + \mathbf{i})(1 - \mathbf{i})$, porém a propiedade da fatoração única se estende aos elementos associados aos fatores, logo 2 também pode ser fatorado na forma $(-1 - \mathbf{i})(-1 + \mathbf{i})$. Essa forma consiste apenas na multiplicação pela unidade -1 dos fatores e não altera ao resultado final. O mesmo poderá ser obtido por qualquer outro elemento associado em qualquer outra fatoração.

Além desses resultados apresentados acima outros ainda nos são necessários para a realização de uma criptografia RSA Gaussiana, como uma aritmética modular gaussiana e teorema análogos ao teorema chinês do resto e ao Teorema de Fermat. No próximo capítulo iremos discutir sobre a viabilidade ou não do algoritmo RSA Gaussiano, além de conhecer alguns resultados relacionados a área pelo ponto de vista de outros pesquisadores do mesmo algoritmo.

CONSIDERAÇÕES FINAIS

Nesta sessão faremos as últimas considerações sobre a viabilidade do algoritmo RSA Gaussiano. Além disso vamos ver o que outros pesquisadores já estão concluindo em suas pesquisas.

A primeira coisa que devemos prestar atenção é que no decorrer deste artigo não encontramos nada que impedisse a realização de uma criptografia RSA Gaussiana, mas como foi visto no capítulo 3, ainda faltam a compravação de alguns teoremas matemáticos importantes para a realização da criptografia RSA Gaussiana.

O material publicado por [KV08] e [EKHAD05] nos leva a crer na viabiliadade do algoritmo. O que ocorre é que ambos possuem visões bem diferentes. [KV08] não defende o algoritmo, pois acredita que ele não acrescenta segurança ao algoritmo RSA, além de deixá-lo menos prático. Abaixo citamos o trecho onde isso é afirmado:

> "The extension of RSA algorithm into the field of Gaussian integers [...] is viable only if real primes p congruent to 3 modulo 4 are used [...]. The extended algorithm could add security only if breaking the original RSA is not as hard as factoring. Even in this case, it is not clear whether the extended algorithm would increase security. The Gaussian integer RSA is slightly less efficient than the original, therefore the original real integer RSA is more practical."

Enquanto isso, [EKHAD05] defende o algoritmo Gaussiano por aumentar a segurança comparado ao clássico, como pode ser lido abaixo:

> "Arithmetic needed for the RSA cryptosystem in the domains of Gaussian integers and polynomials over finite fields were modified and computational procedures were described. There are advantages for the new schemes over the classical one. First, generating the odd prime numbers in both the classical and the

> modified methods requires the same amount of efforts. Second, the modified method provides an extension to the range of chosen messages and the trials will be more complicated. ”

Baseado nos textos de ambos podemos concluir que além da realização de tal algoritmo, outro problema a ser investigado em um trabalho futuro consiste na análise de segurança e complexidade do algoritmo, visto que ainda não possuímos uma conclusão definitiva sobre isso.

REFERÊNCIAS BIBLIOGRÁFICAS

[Con08] Keith Conrad. The gaussian integers. *Pre-Print, paper edition*, 2008.

[Cou07] Severino Collier Coutinho. *Criptografia*. Editora IMPA/SBM, 2007. Coleçã de livros de Iniciação Científica da OBMEP.

[Cou14] Severino Collier Coutinho. *Números Inteiros e Criptografia RSA*. Editora IMPA, 2014.

[DH76] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[EKHAD05] Abdul Nasser El-Kassar, Ramzi A Haraty, YA Awad, and Narayan C Debnath. Modified rsa in the domains of gaussian integers and polynomials over finite fields. In *CAINE*, pages 298–303, 2005.

[Gau15] Carl Friedrich Gauss. *Methodus nova integralium valores per approximationem inveniendi*. Apvd Henricvm Dieterich, 1815.

[KV08] Aleksey Koval and Boris S Verkhovsky. Analysis of rsa over gaussian integers algorithm. In *Information Technology: New Generations, 2008. ITNG 2008. Fifth International Conference on*, pages 101–105. IEEE, 2008.

[Pim06] Elaine Gouvêa Pimentel. Teoria dos números e criptografia rsa. Notas de aula, 2006.

[PR13] Daniel Campolina Pacci and Camila Takeuti Vaz Rodrigues. *Inteiros De Gauss*. 2013.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[SF09] Abraham Sinkov and Todd Feil. *Elementary cryptanalysis*, volume 22. MAA, 2009.

[Sha49] Claude E. Shannon. Communication theory of secrecy systems. *Bell system technical journal*, 28(4):656–715, 1949.