

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**

28 de fevereiro de 2017

Resumo

O presente artigo expõe o resultado da pesquisa para TCC sobre o algoritmo de criptografia RSA gaussiano.

Sumário

1	Introdução	2
2	Primos e Fatoração	7
2.1	Ciclos e Restos	7
2.2	Números Primos e Compostos	7
2.3	Fatoração	8
3	Operações Modulares	10
3.1	Definição de módulo	10
3.2	Propiedades das congruências	11
4	Inversos Modulares	13
4.1	Inversos modulares	13
4.2	Inexistência e existência de inversos	14
	Bibliografia	15

Capítulo 1

Introdução

O sigilo sempre foi uma arma explorada pelos seres humanos para vencer certas batalhas, mesmo que na cotidiana missão de se comunicar. Foi a partir dessa necessidade que se criou o que chamamos de *criptografia*, nome dado ao conjunto de técnicas usadas para se falar e escrever em códigos. Seu objetivo é garantir que apenas as pessoas envolvidas na comunicação possam compreender a mensagem codificada (ou criptografada), 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, para fins pedagógicos iremos introduzir tais conceitos apresentando um dos primeiros algoritmos criptográficos que se tem conhecimento, a criptografia de César, além de seus sucessores.

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

Uma forma muito natural de se generalizar o algoritmo de César é fazer a troca da letra da mensagem por outra em uma posição qualquer fixada. A chamada *criptografia de substituição monoalfabética* consiste em substituir cada letra pela a que ocupa

n posições a sua frente, sendo que o número n é conhecido apenas pelo emissor e pelo receptor da mensagem. Chamamos este número n de *chave criptográfica*. Para podermos compreender a mensagem, precisamos substituir as letras que formam a mensagem criptografado pelas as 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
↓	↓	↓	↓	↓	↓	↓	
$S(+19)$	$E(+5)$	$N(+14)$	$H(+8)$	$A(+1)$	$S(+19)$	$E(+5)$	
↓	↓	↓	↓	↓	↓	↓	Chave
$T(20)$	$G(7)$	$C(29)$	$J(10)$	$P(16)$	$K(37)$	$F(6)$	
							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. Observe que 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 nesse 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.

Durante a Primeira Guerra Mundial o contato físico para a criação de chaves era complicado, isso levou a criação de máquinas automáticas de criptografia. O *Enigma* foi uma destas 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 alemães devido a facilidade de uso e sua suposta indecifrábilidade 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 haja 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 as 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 de mensagens e seu receptor. Com a assimetria de chaves não era mais necessário um contato tão próximo entre emissor e receptor, que já havia sido problema no passado. 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 assimétrica é o *RSA*(RIVEST et al, 1983) [RSA78], algoritmo desenvolvido por Rivest, Shamir e Adleman. Este algoritmo está presente em muitas aplicações de alta segurança, como bancos, sistemas militares e servidores de internet, e ele utiliza para a geração de chaves dois números primos de grandeza superior a 2^{512} multiplicados entre si.

Neste trabalho será feita a exposição detalhada da chamada criptografia RSA clássica, enfatizando a parte matemática relacionada a teoria dos números, necessária para a construção do algoritmo.

O maior objetivo deste é analisar a viabilidade em se propor um criptografia derivada do RSA, mas centrada no conjunto dos números primos de Gauss, que são todos os números complexos de forma $a + bi$ tais que a e b existam no conjunto dos inteiros e $a^2 + b^2$

resulte em um número primo, a qual chamamos de criptografia RSA gaussiana.

Como primeiro avanço necessário para este novo algoritmo se faz necessária a adaptação de uma série de resultados próprios dos números primos para os números primos gaussianos, por isso grande parte do trabalho se centrará em demonstrar resultados matemáticos e discutir algumas dificuldades em obtê-los.

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

Capítulo 2

Primos e Fatoração

2.1 Ciclos e Restos

Para podermos compreender a aritmética modular, precisamos começar entendendo o conceito de ciclicidade, que são os fatos que ocorrem sempre após um determinado período constante. Um bom exemplo deste conceito é o nascer do sol, que é um evento que ocorre sempre após um ciclo de 24 horas, assim como o dia de seu aniversário ocorre uma vez a cada ciclo de um ano.

O mesmo tipo de evento é observado com o resto dos números inteiros. Tomemos por exemplo os restos de divisão pelo número inteiro 4:

<i>Inteiro</i>	1	2	3	4	5	6	7	8	9	10	11	12
<i>Resto</i>	1	2	3	0	1	2	3	0	1	2	3	0

É visível que após 4 números o resto tende a se repetir. O mesmo feito ocorre a qualquer número inteiro n , onde o ciclo se repetirá sempre a cada n iterações. Os números que apresentam o resto 0 são conhecidos como múltiplos de n .

2.2 Números Primos e Compostos

Existe um tipo especial de número que só é múltiplo, ou seja, possui resto 0, em duas condições, quando n é igual a 1 ou quando ele é igual a n . A esse conjunto de números atribui-se o nome de *números primos*.

Existem infinitos números primos, caso não acredite vamos supor que o conjunto finito de primos seja composto por p_1, p_2, \dots, p_r . Considerando que o número inteiro $n = (p_1)(p_2)\dots(p_r) + 1$. n deve possuir um fator p , que está contido em p_1, p_2, \dots, p_r , mas isso significa q p divide 1, o que é absurdo e prova que o conjunto não tem fim.

Todo o número que não é primo é chamado de *Número Composto*, sendo que este número composto pode ser escrito em uma combinação única de fatores primos. O processo de se descobrir estes fatores é chamado de *fatoração*.

2.3 Fatoração

Anteriormente falamos que todo o número pode ser escrito por uma combinação de fatores primos, neste capítulo vamos abordar como se pode obter estes fatores.

Começamos por escolher o número inteiro n ao qual iremos fatorar, em seguida testamos a sua divisibilidade por 2, se for tente dividi-lo novamente por 2, senão passa-se para o próximo número primo, o 3. Repete-se esse procedimento até chegarmos a \sqrt{n} , caso não achemos nenhum fator primo até \sqrt{n} , n é primo.

Quando acabamos de realizar a fatoração, chegamos a um número fatorado da forma $n = (2^{a_1})(3^{a_2})\dots(p^{a_p})$, todo o número inteiro pode ser escrito nessa forma, chamada forma fatorada, veja, por exemplo o $12 = (2^2)(3^1)$ e o $19 = (19^1)$.

Essa forma fatorada nos é formalmente apresentada pelo *Teorema da Fatoração Única*. Ele nos diz que dado um número inteiro $n \geq 2$ pode-se escrevê-lo de forma única como:

$$n = (p_1^{e_1})\dots(p_k^{e_k})$$

onde $1 < p_1 < \dots < p_k$ são primos e e_1, \dots, e_k são inteiros.

Mesmo algoritmo da fatoraão sendo tao simples de se compreender, ele e demorado ate para os mais modernos computadores. Para se ter uma ideia disto, um computador comum executa cerca de 50 divisoes por segundo, para se calcular com certeza que um numero proximo a 10^{100} e primo ele levaria cerca de 317 decilhoes de anos.

Capítulo 3

Operações Modulares

3.1 Definição de módulo

Já lhe foi apresentado anteriormente o conceito da ciclicidade para a definição de restos, neste capítulo iremos nos aprofundar mais sobre esse conceito, estudando as propriedades necessárias da aritmética modular para a elaboração da criptografia RSA.

Um dos conceitos mais importantes da aritmética modular é o de congruência, representado pelo símbolo \equiv . Talvez o exemplo mais comum de congruência em nossas vidas sejam os dias da semana, embora o número do dia venha a variar ao longo do mês, sempre após 7 dias voltará a ser domingo, por exemplo, logo a semana é uma congruência de módulo 7.

Para exemplificar vamos supor que primeiro domingo deste mês foi dia 4, e o último será dia 25, logo temos que

$$4 \equiv 25(mod 7)$$

Fique claro que o que tornou 25 congruente a 4 no módulo 7 não foi o fato de caírem no mesmo dia, isso é apenas um consequência, o que os torna congruentes é o fato de que divididos pelo módulo, no caso 7, eles apresentam o mesmo resto. Esse fato não se repete, por exemplo, se o módulo for 5, neste caso $4 \equiv 4(mod 5)$ e $25 \equiv 0(mod 5)$.

3.2 Propiedades das congruências

Assim como as igualdades e desigualdades, as congruências também possuem uma listagem de propriedades em suas operações. Ao longo desta seção lhe serão demonstradas essas propriedades. Fique atento pois as propriedades das congruências nos facilitarão a compreensão de alguns conceitos importantes do algoritmo RSA mais a frente.

A primeira propriedade das congruências, e a mais simples dela, é a *reflexiva*, onde se diz que um número sempre é congruente a si mesmo. Para termos certeza vamos tomar um número qualquer a , sendo $a \equiv a(modn)$, é equivalente dizermos que $a - a \equiv 0(modn)$. Por 0 ser múltiplo de qualquer número podemos confirmar que $a \equiv a(modn)$.

A propriedade *simétrica* nos diz que se $a \equiv b(modn)$, $b \equiv a(modn)$. A afirmação anterior pode ser escrito como se $a - b$ é múltiplo de n , mas para isso deve ocorrer algum número k que equivala à:

$$a - b = k \cdot n$$

Caso multipliquemos esta equação por -1 , vamos obter:

$$b - a = (-k) \cdot n$$

Que nos prova que $b - a$ é múltiplo de n , logo $b \equiv a(modn)$.

A terceira propriedade das congruências é a *transitiva*, onde se diz que se $a \equiv b(modn)$ e $b \equiv c(modn)$, $a \equiv c(modn)$. Para prová-la vamos observar as equações

$$a - b = k \cdot n \quad e \quad b - c = l \cdot m$$

Sabendo que k e l são inteiros escolhidas de forma adequada as equações, podemos somar as equações, resultando em:

$$(a - b) + (b - c) = k \cdot n + l \cdot m$$

Que pode ser simplificada em:

$$a - c = (k + l) \cdot m$$

Essa equação equivale em valor a $a \equiv c(mod n)$, logo a propriedade transitiva é válida.

Capítulo 4

Inversos Modulares

4.1 Inversos modulares

Nosso objetivo com o decorrer deste capítulo é o de explicar a operação matemática mais importante para para o algoritmo RSA. Para podermos compreendê-la vamos relembrar do conceito ensinado no colégio de inverso multiplicativo, que consiste em obter o número que multiplicado a um número n qualquer resulte em 1. A operação do inverso modular parte do mesmo princípio.

Vamos supor que queremos obter o inverso modular de 6 para o módulo 7, o que nós teremos que fazer então é encontrar qual o número que multiplicado por 6 tem resto 1 quando dividido por 7. Começamos pelo 1, teremos que $6 \cdot 1 = 6$, $6 \equiv 6(mod7)$. Com 2 o resultado será 12, logo $12 \equiv 5(mod7)$, que para nós também não serve. Tentando o 3 obtemos 4 e com 4 obtemos 3. Com o 5 nosso retorno será 2. Finalmente quando chegamos ao 6 nós temos que $6 \cdot 6 = 36$, $36 \equiv 1(mod7)$. Com isso podemos concluir que o inverso multiplicativo de 6 no módulo 7 é o próprio 6.

Para simplificar o que foi dito acima, podemos dizer a operação de inverso multiplicativo no módulo n para a consiste em encontrar um número a' tal que:

$$a \cdot a' \equiv 1(modn)$$

4.2 Inexistência e existência de inversos

Antes de começarmos vamos tentar calcular o inverso multiplicativo de 2 no módulo 8, vamos lá:

$$\begin{aligned}2 \cdot 0 &\equiv 0 \not\equiv 1(\text{mod}8) \\2 \cdot 1 &\equiv 2 \not\equiv 1(\text{mod}8) \\2 \cdot 2 &\equiv 4 \not\equiv 1(\text{mod}8) \\2 \cdot 3 &\equiv 6 \not\equiv 1(\text{mod}8) \\2 \cdot 4 &\equiv 8 \equiv 0 \not\equiv 1(\text{mod}8) \\2 \cdot 5 &\equiv 10 \equiv 2 \not\equiv 1(\text{mod}8) \\2 \cdot 6 &\equiv 12 \equiv 4 \not\equiv 1(\text{mod}8) \\2 \cdot 7 &\equiv 14 \equiv 6 \not\equiv 1(\text{mod}8)\end{aligned}$$

Não encontramos nenhuma resposta pois, simplesmente, não há. Antes que se pergunte o motivo de não tentarmos com números maiores que 7, é válido lembrar que a partir do 8 teríamos a repetição de resultados por conta das congruências.

A operação de inverso multiplicativo só possui resultado em casos onde o número a ao qual queremos calcular o inverso e o módulo são *primos entre si*, ou seja, não possuam nenhum fator em comum. Por conta disso usamos os números primos no algoritmo RSA.

Para comprovar o que foi dito acima, vamos tomar um número a , tal que

$$a \cdot a' \equiv 1(\text{mod}n)$$

Isso pode ser traduzido em linguagem humana como n divide $a \cdot a' - 1$. Isso em linguagem matemática pode ser escrito como:

$$a \cdot a' - 1 = n \cdot k$$

Como estamos atrás de saber se a e n não possuem fator comum, então há de haver um k inteiro para a equação acima. Nosso primeiro passo para provar isso será de se criar o conjunto $V(a, n)$, esse conjunto é formado por inteiros positivos e pode ser escrito como

$$x \cdot a + y \cdot n$$

Referências Bibliográficas

- [DH76] Diffie, Whitfield e Martin Hellman: New directions in cryptography. IEEE transactions on Information Theory, 22(6):644–654, 1976.
- [Gau15] Gauss, Carl Friedrich: Methodus nova integralium valores per approximationem inveniendi. apvd Henricvm Dieterich, 1815.
- [mil] Millennium Problems — Clay Mathematics Institute.
<http://www.claymath.org/millennium-problems>. Acessado em 15/11/2016.
- [Rie59] Riemann, Bernhard: Ueber die Anzahl der Primzahlen unter einer gegebenen Grosse. Ges. Math. Werke und Wissenschaftlicher Nachlaß, 2:145–155, 1859.
- [RSA78] Rivest, Ronald L, Adi Shamir e Leonard Adleman: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21(2):120–126, 1978.
- [SF09] Sinkov, Abraham e Todd Feil: Elementary cryptanalysis, volume 22. MAA, 2009.
- [Sha49] Shannon, Claude E: Communication theory of secrecy systems. Bell system technical journal, 28(4):656–715, 1949.