https://eventos.utfpr.edu.br//sicite/sicite2020

Criptografia RSA e o método de Pollard de fatoração de inteiros

RSA encryption and Pollard's method for factorization of integers

Camila Beatriz da Silva
csilva.2000@alunos.utfpr.edu.br,
Universidade Tecnológica Federal do Paraná, Apucarana, Paraná, Brasil

Rodrigo dos Santos Veloso Martins
rodrigomartins@utfpr.edu.br
Universidade Tecnológica Federal do Paraná, Apucarana, Paraná, Brasil

RESUMO

A presente pesquisa trata-se de um estudo sobre a criptografia RSA, um tipo de criptografia de chave pública. A pesquisa traz uma revisão as fases características do sistema criptográfico RSA (pré-codificação, codificação e decodificação) e uma implementação em linguagem C do sistema. Foi realizada também uma implementação em linguagem C de uma etapa de um ataque ao RSA, onde o Método de Pollard é utilizado na fatoração da chave pública. Os resultados mostram que a função e o ponto inicial escolhidos no Método de Pollard interferem no método e número de iterações e o tempo de execução crescentes indicam a complexidade em fatorar números maiores.

PALAVRAS-CHAVE: Criptografia. RSA. Fatoração de inteiros.

ABSTRACT

This research is a study of the RSA cryptosystem, which is a publick-key system. We review the characteristics phases of the system (pré-codification, codification and decodification) and na implementation in C language of the cryptosystem. A step in na attack to the system was also implemented in C language, where Pollard's Method is used in the factorization of the public key. The results show that the function and initial point chosen in Pollard Method have na impact on the method.

KEYWORDS: Cryptography. RSA. Factorization of integers.

Recebido: 19 ago. 2020.
Aprovado: 01 out. 2020.

INTRODUÇÃO

A criptografia é uma área da matemática e da ciência da computação que visa tornar comunicações digitais seguras, por exemplo, ao tornar uma mensagem que necessita ser secreta em algo ilegível para qualquer pessoa que não seja o destinatário. Esta área possui grande relevância na atualidade, devido principalmente à necessidade de garantir segurança do volume cada vez maior de dados que trafega na Internet, em especial em setores que lidam com questões financeiras, como o setor bancário (Internet Banking) e o setor de comércio eletrônico.

Entre as diversas maneiras existentes de criptografar uma mensagem, a criptografia RSA é usada há muito tempo e foi criada em 1978 por R.L. Rivest, A. Shamir e L. Adleman. O RSA é um tipo de criptografia de chave pública, o que quer dizer que todos os usuários possuem uma chave pública e uma chave privada, tanto o remetente quanto o destinatário da mensagem. O remetente usará a chave pública do destinatário para criptografar a mensagem. Já a chave privada é aquela que o destinatário possui para decriptar a mensagem recebida. A decriptação só é possível obtendo essa chave privada e, como somente o destinatário possui a sua chave, somente ele poderá decriptá-la.

MATERIAL E MÉTODOS

No início do processo de encriptação de uma mensagem, existe a necessidade de realizar uma pré-codificação, ou seja, transformar a mensagem em números (COUTINHO, 1997). Isto pode ser feito letra a letra, nesse caso há a necessidade de encontrar correspondentes numéricos para cada caracter, podendo ser utilizada, por exemplo, a tabela American Standard Code for Information Interchange (ASCII), Figura 1. Consideramos nesta seção que as mensagens enviadas utilizam um alfabeto de 27 letras, o que corresponde ao alfabeto usual acrescido de um caracter para o espaço em branco. O conjunto de possíveis correspondentes numéricos de bloco de mensagem é {0, 1, ..., 26}.

Figura 1 – Tabela ASCII

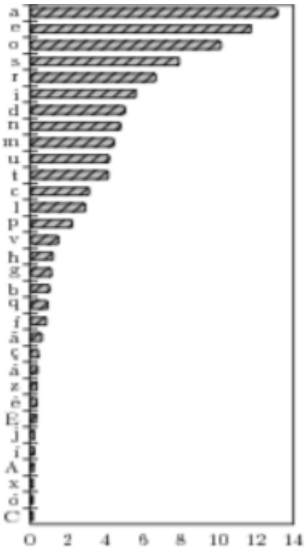
ASCII control characters			ASCII printable characters						Extended ASCII characters							
00	NULL	(Null character)	32	space	64	@	96	`	128	Ç	160	á	192	└	224	Ó
01	SOH	(Start of Header)	33	!	65	A	97	a	129	ü	161	í	193	┴	225	ß
02	STX	(Start of Text)	34	"	66	B	98	b	130	é	162	ó	194	┬	226	Ô
03	ETX	(End of Text)	35	#	67	C	99	c	131	â	163	ú	195	├	227	Ò
04	EOT	(End of Trans.)	36	$	68	D	100	d	132	ä	164	ñ	196	─	228	õ
05	ENQ	(Enquiry)	37	%	69	E	101	e	133	à	165	Ñ	197	┼	229	Õ
06	ACK	(Acknowledgement)	38	&	70	F	102	f	134	å	166	ª	198	ã	230	µ
07	BEL	(Bell)	39	'	71	G	103	g	135	ç	167	º	199	Ã	231	þ
08	BS	(Backspace)	40	(	72	H	104	h	136	ê	168	¿	200	╚	232	Þ
09	HT	(Horizontal Tab)	41	)	73	I	105	i	137	ë	169	®	201	╔	233	Ú
10	LF	(Line feed)	42	*	74	J	106	j	138	è	170	¬	202	╩	234	Û
11	VT	(Vertical Tab)	43	+	75	K	107	k	139	ï	171	½	203	╦	235	Ù
12	FF	(Form feed)	44	,	76	L	108	l	140	î	172	¼	204	╠	236	ý
13	CR	(Carriage return)	45	-	77	M	109	m	141	ì	173	¡	205	═	237	Ý
14	SO	(Shift Out)	46	.	78	N	110	n	142	Ä	174	«	206	╬	238	¯
15	SI	(Shift In)	47	/	79	O	111	o	143	Å	175	»	207	¤	239	´
16	DLE	(Data link escape)	48	0	80	P	112	p	144	É	176	░	208	ð	240	≡
17	DC1	(Device control 1)	49	1	81	Q	113	q	145	æ	177	▒	209	Ð	241	±
18	DC2	(Device control 2)	50	2	82	R	114	r	146	Æ	178	▓	210	Ê	242	‗
19	DC3	(Device control 3)	51	3	83	S	115	s	147	ô	179	│	211	Ë	243	¾
20	DC4	(Device control 4)	52	4	84	T	116	t	148	ö	180	┤	212	È	244	¶
21	NAK	(Negative acknowl.)	53	5	85	U	117	u	149	ò	181	Á	213	ı	245	§
22	SYN	(Synchronous idle)	54	6	86	V	118	v	150	û	182	Â	214	Í	246	÷
23	ETB	(End of trans. block)	55	7	87	W	119	w	151	ù	183	À	215	Î	247	¸
24	CAN	(Cancel)	56	8	88	X	120	x	152	ÿ	184	©	216	Ï	248	°
25	EM	(End of medium)	57	9	89	Y	121	y	153	Ö	185	╣	217	┘	249	¨
26	SUB	(Substitute)	58	:	90	Z	122	z	154	Ü	186	║	218	┌	250	·
27	ESC	(Escape)	59	;	91	[	123	{	155	ø	187	╗	219	█	251	¹
28	FS	(File separator)	60	<	92	\	124	|	156	£	188	╝	220	▄	252	³
29	GS	(Group separator)	61	=	93	]	125	}	157	Ø	189	¢	221	¦	253	²
30	RS	(Record separator)	62	>	94	^	126	~	158	×	190	¥	222	Ì	254	■
31	US	(Unit separator)	63	?	95	_			159	ƒ	191	┐	223	▀	255	nbsp
127	DEL	(Delete)														

Fonte: Disponível em: https://theasciicode.com.ar/. Acesso: 29 out. 2020

Um outro método de pré-codificação utiliza blocos de duas letras, também conhecidos como dígrafos. A criptografia com dígrafos é semelhante àquela realizada letra por letra, mas com a diferença que o bloco de mensagem está entre $27^2 = 729$ possibilidades A pré-codificação com dígrafos é feita a partir de uma função definida como $b = 27x + y$, sendo x a primeira letra e y a segunda letra do dígrafo e b o correspondente numérico do dígrafo que será encriptado, pertencente ao intervalo {0, 1,..., 728} (KOBLITZ, 1994).

A pré-codificação com dígrafos apresenta maior a dificuldade de decriptação que a pré-codificação simples. Por exemplo, se um adversário consegue interceptar uma mensagem que foi cifrada caracter a caracter, utiliza-se a análise de frequência de cada letra (Figura 2). Essa análise é feita baseada na frequência que cada letra é encontrada em um conjunto amplo de textos de uma dada língua. Quando ocorre a encriptação de uma mensagem, as informações de um determinado caracter ficam inalteradas, apenas implícitas em outro caracter, como ocorre com a Cifra de César com chave 3, o qual o caracter “a” torna-se o caracter “d” (KOBLITZ, 1994).

Figura 2 – Frequência Relativa das Letras no Português



Fonte: (Quaresma, Pinho, 2019).

Após realizar a etapa de pré-codificação, inicia-se a codificação. Supomos que Ana deseja enviar uma mensagem criptografada para Bob no sistema RSA. Bob necessita criar sua chave pública para que Ana possa utilizá-la para criptografar a mensagem, logo ele deve escolher dois números primos distintos denotados por p e q. Bob pode então calcular $\Phi(n)$, conhecida como Função Totiente de Euller, onde $\Phi(n)$ = {n em Z: $MDC(e, \Phi(n)) = 1$}. No caso em que $n = p * q$, temos:

$$\Phi(n) = (p - 1)(q - 1). \quad (1)$$

Bob irá precisar também de e, um número inteiro positivo que seja inversível módulo $\Phi(n)$. Um número é inversível $mod\, \Phi(n)$ se não tiver divisor comum com Φ. Seja d o seu inverso módulo $\Phi(n)$. Podemos chamar essa etapa de Criação de Chaves. Cada pessoa possui seu número n, sendo ele público e p e q mantidos privados, pois caso um adversário fatore o número n, obtendo, assim,

os primos p e q, assim, a segurança do RSA é quebrada. A chave pública de Bob é (n, e) e a chave privada (n, d).

A partir da pré-codificação, a mensagem foi separada em blocos, desta forma, Ana necessita do par (n, e), codificando um bloco por vez e ao final existirá uma sequência de blocos codificados. Um bloco b encriptado é denotado por $C(b)$ e irá ser codificado como:

$$\mathrm{C(b)} = \mathrm{b^e mod(n)}. \quad (2)$$

O processo inverso, no qual Bob recebe a mensagem e deve decriptar é denominado decodificação, e precisará da chave privada (n, d), resultando na decriptação do bloco $a = C(b)$, sendo denotada por $D(a)$:

$$D(a) = a^d mod(n), \quad (3)$$

onde a é o bloco $C(b)$ encriptado e d é o inverso de e módulo $\Phi(n)$. É possível provar que D(C(b)) = b (COUTINHO, 1997).

Foi realizada em linguagem C uma implementação do sistema RSA começando com a declaração das variáveis p e q, atribuindo valores primos a elas. Esta atribuição pode ser feita através de testes de primalidade, mas no caso da implementação feita os primos foram escolhidos manualmente a partir de listas conhecidas de primos. Logo após declara-se as variáveis: n, a multiplicação dos primos p e q; Φ, como na Eq. (1); e b, o bloco a ser cifrado já feito sua pré-codificação.

Posteriormente, implementa-se uma função para achar o número e, chamando uma função já declarada denotada por *Greatest Common Divisor* (GCD). Calcula-se o mdc entre e e $\Phi(n)$: devemos ter $MDC(e, \Phi(n)) = 1$.

Em seguida, para encontrar d, é calculado o inverso $d = e^{-1} \, mod \, \Phi(n)$ pelo *Algoritmo Euclidiano* Estendido, com uma função chamada mul_inv retornando o inverso de e módulo $\Phi(n)$, isto é, $mul\_inv(e, \Phi(n)) = e^{(-1)} \, mod \, \Phi(n)$.

E por fim, a declaração de duas funções primordiais, a primeira chamada Criptografar possuindo como entrada as variáveis b, n e e, e armazenando a encriptação da mensagem em c. Dentro da função é aberto o arquivo o qual consta a mensagem a ser encriptada, logo em seguida é feito um for de 1 até e, com c inicializando em 1, executa-se em cada iteração:

$$c = (c * b) \,\%\, n, \quad (4)$$

onde $(c * b) \,\%\, n = (c * b) \, mod \, n$. Este processo resulta na codificação de b, ou seja, $C(b)$, como na Eq. (2).

A segunda função, chamada Decriptografar, possui as variáveis de entrada c, n e d, e saída b. Sendo somente o inverso da função acima, foi feito um for de 1 até d, e b inicializando em 1, executando em cada iteração:

$$b = (c * b) \,\%\, n, \quad (5)$$

efetuando a decriptação de $D(c)$, como na Eq. (3). Ao final, utilizando um if, caso $b < 0$, soma-se apenas n.

Para criar um sistema criptográfico, matemáticos e criptógrafos necessitam realizar vários testes para chegar à conclusão de que esse sistema proposto é

seguro. Para isso, utilizam dos melhores algoritmos disponíveis para quebrar o criptosistema. Para passar tranquilidade às empresas que utilizam de criptografia como forma de segurança para seus dados, existem organismos de normalização que avaliam e recomendam o uso de criptosistemas aprovados (KOBLITZ; MENEZES, 2004). Exemplos de órgãos são: ANSI, IEEE, ISO e o NIST. Por exemplo, o NIST (*National Institute of Standards and Technology*, EUA) recomenda no documento FIPS 186-4 de 2013 que o módulo n do RSA para assinatura digital seja maior ou igual a 1024 bits (KERRY; DIRECTOR, 2013.). Este será substituído pelo FIPS 186-5, ainda em fase de "draft" e recebendo comentários públicos. O FIPS 186-5, de 2019, recomenda um módulo de pelo menos 2048 bits para o RSA.

As maneiras de ataques ao RSA, não se restrigem apenas a calcular o inverso da função usada por ele, ou a fatoração do inteiro n. Por exemplo, métodos baseados em escolhas inapropriadas de d e e e os denominados $side-channel\ attacks$, que utilizam propriedades específicas da implementação e operação da máquina para decriptar a mensagem, como tempo de execução, consumo de energia, radiação eletromagnética, podem ocasionar uma ruptura na segurança (KOBLITZ; MENEZES, 2004).

Visto que a segurança do RSA pode ser quebrada a partir da fatoração de inteiros, temos como exemplo o Método de Pollard (POLLARD, 1975.). Este método consiste em escolhermos x_0, ou seja, um número representando o ponto inicial dos nossos cálculos, e uma função, esta é reduzida módulo n, o qual é o número que queremos fatorar, e uma sequência de pontos é gerada:

$$x_{i+1} \equiv x_i^2 - 1\ mod(n). \tag{6}$$

Forma-se assim a tripla de principais variáveis sendo elas (x_i, x_{2i}, Q_i), com $i = 1, 2, 3, \ldots$, onde temos:

$$Q_i \equiv \Pi_{j=1}^{i} (x_{2j} - x_j)\ mod(n). \tag{7}$$

E para finalizar uma iteração deste método, calculamos o maior divisor comum entre Q_i e n, e quando o $1 < mdc < n$ temos um divisor não trivial de n. A função utilizada foi $y = x^2 - 1$, no caso, a função pode ser substituída por outra, porém cada função exerce uma dinâmica diferente.

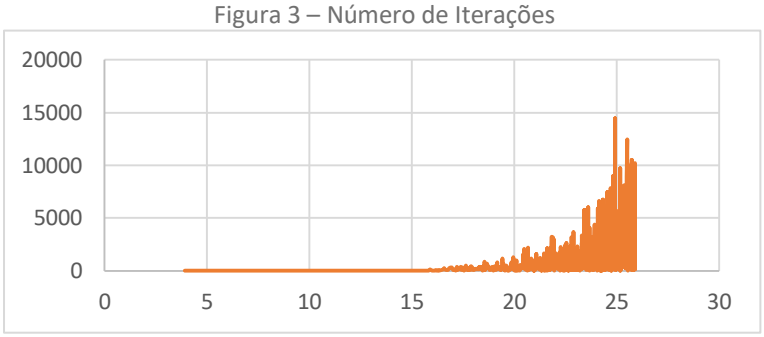
Foi realizado um experimento com base nesse Método, utilizando uma função Tempo para calcular em quantos milissegundos cada fatoração duraria. Foi calculado quantas iterações foram necessárias para fatorar determinado número e se o número realmente foi fatorado ou não, tendo como ponto de partida $x_0 = 7$, e com a mesma função proposta. Este experimento foi realizado com os 1000 primeiros números primos, em cada execução do método foram usados números primos p e q de uma lista de primos, onde p está no intervalo [2, 7907] e q era definido como o número primo seguinte a p.

O método de Pollard foi implementado em linguagem C, em um notebook Windows 10 64 bits, com processador Intel i7-8550U 1.80GHz, compilador 6.3.0 e 16GB de memória RAM.

RESULTADOS E DISCUSSÃO

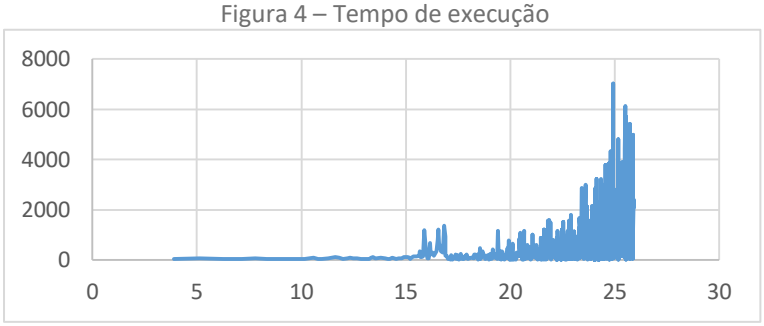
Nas Figuras 3 e 4 são apresentados, respectivamente, no eixo vertical, o número de iterações e o tempo de execução (em milissegundos) para cada execução do método. No eixo horizontal consta o valor de $log_2(n)$ para cada $n = p * q$. A escolha de $log_2(n)$ no eixo horizontal se deu pelo fato de que a entrada de um de fatoração é considerada usualmente como o comprimento da representação binária do inteiro n (MONTGOMERY, 1994).

Figura 3 – Número de Iterações



Fonte: Autoria própria.

Figura 4 – Tempo de execução



Fonte: Autoria própria.

Foi feito um cálculo da quantidade de números n que foram fatorados com sucesso, e 79,2% dos 999 números foram fatorados. O restante não obteve sucesso e o problema poderia ser resolvido ao escolher uma nova função e um novo ponto de partida para uma nova execução.

CONCLUSÃO

A função e o ponto de partida para os cálculos podem afetar a quantidade de interações ou no tempo do Método de Pollard para cada n, esse fenômeno pode ser observado nas oscilações presentes nos gráficos. Porém apesar das oscilações, as curvas crescentes indicam a dificuldade de se fatorar rapidamente um números inteiros maiores.

REFERÊNCIAS

COUTINHO, S. C. **Números inteiros e criptografia RSA.** 2. ed. Rio Janeiro: IMPA, 2014.

KERRY, C. F.; SECRETARY, A., DIRECTOR, C. R. **FIPS PUB 186-4** Federal information processing standards publication Digital Signature Standard (DSS). Gaithersburg: National Institute Of Standards And Technology (NIST), 2013. Disponível em: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf. Acesso em: 29 out. 2020.

KOBLITZ, N.; MENEZES, A. J. A Survey of public-key cryptosystems. **Siam Review**, v. 46, n. 4, p. 599-634, 2004. Disponível em: https://epubs.siam.org/doi/abs/10.1137/S0036144503439190?journalCode=siread. Acesso em: 29 out. 2020.

MONTGOMERY, P. L. A survey of modern integer factorization algorithms. **CWI Quarterly**, v. 7, n. 4, p. 337-366, 1994. Disponível em: https://ir.cwi.nl/pub/18252B.pdf. Acesso em: 29 out. 2020.

POLLARD, J. M. A monte carlo method for factorization. **BIT Numerical Mathematics**, v. 15, n. 3, p. 331-334, 1975. Disponível em: http://www.cs.cmu.edu/afs/cs.cmu.edu/Web/People/avrim/451f11/lectures/lect1122_Pollard.pdf. Acesso em: 29 out. 2020.

QUARESMA, P.; PINHO, A. Análise de frequências da língua portuguesa. *In*: LIVRO DE ACTAS DA CONFERÊNCIA IBERO-AMERICANA INTERTIC, 2007, Porto. **Anais...** Porto: IASK, 2007. Disponível em: http://www.mat.uc.pt/~pedro/lectivos/CodigosCriptografia1213/interTIC07pqap.pdf. Acesso em: 29 out. 2020.