IMPLEMENTAÇÃO DO ALGORITMO RSA

Autor: José Roberto Bollis Gimenez

ABSTRACT

In this tutorial paper it is reviewed the RSA Algorithm and is presented a new approach on the theoretical basis that supports the Algorithm. Some examples are presented, exposing the algebraic structure of the numbers by means of tables in which columns the cipher process can be observed. Complementing, some details concerning the practical implementation of the algorithm are discussed, especially focusing computational issues.

RESUMO

Neste artigo o Algoritmo RSA é revisto, juntamente com uma nova abordagem ao ferramental matemático que o sustenta. Alguns exemplos são apresentados, mostrando a estrutura algébrica dos números por meio de tabelas em cujas colunas o processo criptográfico pode ser entendido. Complementando, é apresentada uma discussão sobre a implementação prática do algoritmo, visando especialmente os aspectos computacionais.

INTRODUÇÃO

O algoritmo RSA foi uma das primeiras soluções para a criptografia de chave pública (ou de chave assimétrica, usando uma outra nomenclatura). Apesar dos contínuos esforços na proposição de novos métodos criptográficos, este algoritmo ainda representa um importante papel na implementação de mecanismos de autenticação e privacidade nos sistemas de comunicação de dados. O atual momento, em que se observa a explosão do comércio eletrônico na Internet, e uma conseqüente demanda por recursos de segurança, constitui uma boa oportunidade para rever a estrutura matemática deste algoritmo e tecer considerações sobre sua implementação prática.

O algoritmo RSA é assim denominado devido ao nome de seus autores, Ronald Rivest, Adi Shamir e Leonard Adleman, cujo trabalho foi primeiramente publicado em abril de 1977 [RIVEST et al]. Este algoritmo tem sido a base dos principais sistemas de segurança comercialmente utilizados na Internet. Ainda que seja discutível o registro de patentes para um procedimento matemático, este algoritmo teve sua patente requerida, redundando inclusive em algumas demandas judiciais ao longo de sua duração [FLINN & JORDAN]. O Algoritmo RSA teve restrições de utilização em favor da empresa RSA até setembro de 2000, com a expiração do registro de patente.

CRIPTOGRAFIA DE CHAVE ASSIMÉTRICA

O sistema de criptografia de chave assimétrica (ou chave pública) recebe este nome em contraposição ao sistema tradicional de chave simétrica (ou chave secreta) em que os dados são criptografados usando-se uma chave conhecida somente pelas entidades envolvidas na transação.

Na criptografia de chave assimétrica a chave utilizada para gerar a informação cifrada não se presta para decodifica-la. Para isso é necessária uma outra chave, que juntamente com a primeira constituem um par, conhecido como chave pública e chave privada. Uma vez que estas chaves possuem funções diferenciadas, a chave pública pode ser distribuída ou divulgada, sem que haja risco de que ela seja usada para a função contrária, que é realizada pela chave privada [SCHNEIER, 1996].

BASE MATEMÁTICA

O Algoritmo RSA utiliza exponenciação para codificar e decodificar a informação. As operações são efetuadas usando álgebra módulo *N*, de forma que operandos e resultados de operações sejam elementos de um conjunto numérico finito. Nesta álgebra os elementos são tomados como sendo o resto da divisão de inteiros por *N*. Dessa forma, operações de soma e multiplicação são possíveis, devendo o resultado ser reduzido pela função módulo para um número pertencente ao conjunto $Z_N = \{0, 1, \ldots N-1\}$.

Antes de abordar o Algoritmo RSA vamos considerar, a título de ilustração, um processo semelhante de encriptação que utiliza a equação:

$$C = M^e \bmod N \tag{1}$$

onde *M* representa a informação que se deseja criptografar e *C* é o texto cifrado através desse processo. Na verdade, o que se faz na prática é tomar a seqüência de bits de uma mensagem, ou de um arquivo digital, e secioná-la em blocos de comprimento *k* bits. O valor de *M*, para cada um destes blocos, é obtido a partir da combinação numérica binária representada pelo bloco. Obviamente, *k* deve ser escolhido de forma que o número de combinações possíveis seja menor ou igual a *N*. Ou seja, $2^k \leq N$. Uma vez que *C* é obtido através da operação módulo *N*, temos que, tanto *M* quanto *C* são elementos do conjunto {*0, 1, . . . N-1*}.

O processo de decriptação é obtido através de um método semelhante ao de encriptação, cuja função é realizar uma operação inversa à que foi utilizada na codificação. Neste caso:

$$M = C^d \ mod \ N \tag{2}$$

Tendo em vista que as equações (1) e (2) devem realizar funções inversas, é preciso que haja uma escolha apropriada de *d* e *e* para que estas operações se processem. Muito embora a utilização de valores altos para *N* seja um fator crucial na implementação de códigos fortes, vamos primeiramente analisar o comportamento das equações (1) e (2) em um campo com um número reduzido de elementos. Tomemos assim a álgebra módulo 11.

Neste campo algébrico, temos que as operações de soma e multiplicação usando a função módulo 11 são bem definidas, conforme ilustra a Tabela 1:

+	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	0	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10	0
2	2	3	4	5	6	7	8	9	10	0	1
3	3	4	5	6	7	8	9	10	0	1	2
4	4	5	6	7	8	9	10	0	1	2	3
5	5	6	7	8	9	10	0	1	2	3	4
6	6	7	8	9	10	0	1	2	3	4	5
7	7	8	9	10	0	1	2	3	4	5	6
8	8	9	10	0	1	2	3	4	5	6	7
9	9	10	0	1	2	3	4	5	6	7	8
10	10	0	1	2	3	4	5	6	7	8	9

.	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	10
2	0	2	4	6	8	10	1	3	5	7	9
3	0	3	6	9	1	4	7	10	2	5	8
4	0	4	8	1	5	9	2	6	10	3	7
5	0	5	10	4	9	3	8	2	7	1	6
6	0	6	1	7	2	8	3	9	4	10	5
7	0	7	3	10	6	2	9	5	1	8	4
8	0	8	5	2	10	7	4	1	9	6	3
9	0	9	7	5	3	1	10	8	6	4	2
10	0	10	9	8	7	6	5	4	3	2	1

Tabela 1 - relações de adição e multiplicação módulo 11
Table 1 - addition and multiplication modulo 11

Deve-se notar que o resultado das operações é sempre um número pertencente ao conjunto {*0, 1, .. N-1*}, comprovando a propriedade de fechamento das operações módulo N sobre este conjunto finito.

Além das operações de soma e multiplicação módulo *N*, é também possível definir uma outra classe de operações, mais interessante para o propósito deste artigo: a operação de exponenciação. A Tabela 2 ilustra todos os possíveis resultados para expoentes compreendidos entre 1 e 11.

As colunas da tabela 2 correspondem exatamente aos valores de *C*, obtidos através de (1), em função do número atribuído ao expoente *e*. Desta tabela podemos tirar várias conclusões interessantes. Primeiramente, devemos observar que nem todos os números são aplicáveis à escolha de *e*. Os números 2, 4, 5, 6, 8 e 10, por exemplo, não se prestam ao processo de codificação, uma vez que diferentes mensagens resultariam em um mesmo texto cifrado. Fazendo-se $e = 4$, por exemplo, seria impossível decifrar a mensagem *M* quando fosse encontrado o texto cifrado $C = 9$, visto que tanto $M = 5$ quanto $M = 6$ poderiam ter gerado este resultado. Dessa forma, os valores viáveis para a escolha de *e* seriam: 3, 7 e 9, em cujas colunas da tabela não ocorre a repetição de nenhum elemento. Note-se que $d = 11$ também atende a esta condição, muito embora neste caso não teríamos uma codificação, uma vez que o texto cifrado seria igual ao original. Outro fato notável na Tabela 2 é que a partir de X^{10} as colunas da tabela passam a se repetir. Esta é uma conseqüência imediata da utilização de um campo finito de elementos. Após o surgimento de uma primeira coluna repetida, como $X^{10} \ mod\ 11 = X^0 \bmod 11$, as demais colunas se repetem sucessivamente, pois cada coluna é resultado da multiplicação da anterior por *X*. De forma geral, para o caso da álgebra módulo 11, temos que $X^i \ mod\ 11 = X^{i-10} \bmod 11$.

X	**X^2**	**X^3**	**X^4**	**X^5**	**X^6**	**X^7**	**X^8**	**X^9**	**X^{10}**	**X^{11}**
1	1	**1**	1	1	1	**1**	1	**1**	1	**1**
2	4	**8**	5	10	9	**7**	3	**6**	1	**2**
3	9	**5**	4	1	3	**9**	5	**4**	1	**3**
4	5	**9**	3	1	4	**5**	9	**3**	1	**4**
5	3	**4**	9	1	5	**3**	4	**9**	1	**5**
6	3	**7**	9	10	5	**8**	4	**2**	1	**6**
7	5	**2**	3	10	4	**6**	9	**8**	1	**7**
8	9	**6**	4	10	3	**2**	5	**7**	1	**8**
9	4	**3**	5	1	9	**4**	3	**5**	1	**9**
10	1	**10**	1	10	1	**10**	1	**10**	1	**10**

Tabela 2 - Exponenciação módulo 11
Table 2 - Exponentiation modulo 11

Para que seja permitida a decriptografia de mensagens, é necessário que sejam atendidas as condições das equações (1) e (2). Substituindo a segunda equação na primeira, temos:

$$M = [M^e \bmod N]^d \bmod N$$

ou

$$M = M^{de} \bmod N$$

Para o caso em questão, este resultado equivale a escolher na Tabela 2 uma coluna X^i que atenda a duas condições: a primeira é que esta coluna seja uma repetição de X^1, ou seja, que $i = 10\ k + 1,\ k = 1, 2, \ldots$; a segunda é que i seja o produto de dois números inteiros (os números d e e). Dessa forma, uma possibilidade para a criação de um sistema de criptografia seria escolher $i = 21$, de onde teríamos os números 3 e 7 para d e e (notar que d e e podem ser intercambiados).

Antes de prosseguir com a descrição deste sistema de criptografia é preciso entender melhor o processo de repetição das colunas da Tabela 2. Como podemos observar, a periodicidade de repetição, nesta tabela, é de $N - 1$. Isto acontece sempre que N for um número primo, como demonstra o teorema a seguir, também conhecido como "Pequeno Teorema de Fermat".

Teorema 1

Seja N um número primo. Então, para qualquer X pertencente a Z_N:

$$X^{N-1} \bmod N = 1$$

ou, de forma equivalente:

$$X^N \bmod N = X \tag{3}$$

Este teorema se prova por indução finita. Supondo que a expressão (3) seja verdadeira para um determinado elemento X, mostra-se que, neste caso, ela é também verdadeira para $X + 1$. Desenvolvendo o primeiro membro de (3), temos:

$$(X+1)^N \bmod N = \left[X^N + \binom{N}{N-1} X^{N-1} + \binom{N}{N-2} X^{N-2} + \ldots + \binom{N}{1} X + 1 \right] \bmod N \tag{4}$$

onde foi usada a definição dos coeficientes binomiais:

$$\binom{N}{k} = \frac{N!}{(N-k)!\,k!}$$

Notando que os estes números são inteiros e múltiplos de *N*, podemos reescrever (4) como:

$$(X+1)^N \bmod N = X^N \bmod N + 1 \qquad N \ primo$$

pois a operação módulo *N* irá anular todos os termos que contenham *N* como fator. E como por hipótese estamos admitindo $X^N \ mod \ N = X$, vem:

$$(X+1)^N \bmod N = X + 1 \qquad N \ primo$$

Vemos então que se a expressão (3) for válida para determinado elemento *X*, ela valerá também para o elemento *X+1*. E como ela vale para o elemento 0, fica demonstrado o Teorema.

Aplicando o Teorema 1, demonstra-se que para *N* primo, o padrão de repetição para a exponenciação módulo *N* ocorre com uma periodicidade *N-1*, como já havíamos observado pelas colunas da Tabela 2. A escolha de *d*, portanto, deve ser tal que:

$$[e\ d] \ mod \ (N-1) = 1 \qquad (5)$$

Embora o método descrito acima permita a criptografia e decriptografia de mensagens e demais informações digitais, ele não se presta para a implementação de um sistema de chave pública, ou chave assimétrica. Isso porque, usando o conceito do sistema de chave pública, haveria a distribuição dos valores *N* e *e*, para que terceiros pudessem criptografar mensagens usando a equação (1). O detentor da chave privada correspondente a esta chave pública, no entanto, reservaria para si o valor de *d*, de modo que apenas ele pudesse aplicá-lo à equação (2) e proceder à decriptografia das mensagens. Entretanto, observamos que, de posse de *e* e *N* é perfeitamente possível obter *d* através de (5) e, dessa forma, ter acesso às relações usadas na decriptação. Ou seja, a chave privada seria facilmente descoberta, pondo em risco a confidencialidade do

sistema. Concluímos então que para a implementação de um sistema confiável de criptografia de chave pública é necessário, além de relações matemáticas bem definidas entre o par de chaves, a existência de dificuldades adicionais que tornem a descoberta destas relações uma tarefa não trivial.

A solução para esta questão, no caso do Algoritmo RSA, é usar para *N* um número não primo, ao invés de um número primo. Veremos que, neste caso, também existe um padrão de repetição no comportamento da função exponencial módulo *N*, o qual pode ser obtido matematicamente e empregado na determinação dos números *e* e *d* para serem usados nas chaves pública e privada. Entretanto, este parâmetro não é um resultado simples, como *N-1*, no caso de *N* primo, o que dificulta a descoberta da chave privada.

O Algoritmo RSA usa para o valor de *N* um número gerado pela multiplicação de dois números primos, *p* e *q*. Para melhor ilustrar este fato, vamos analisar um exemplo usando *N = 15*, onde *p* e *q* são, conseqüentemente, os números 3 e 5. Do mesmo modo feito acima, para o caso de *N* primo, podemos montar a tabela 3, cujas colunas representam a codificação resultante da escolha dos diferentes valores para *e* na equação (1).

X	**X^2**	**X^3**	**X^4**	**X^5**	**X^6**	**X^7**	**X^8**	**X^9**	**X^{10}**	**X^{11}**	**X^{12}**	**X^{13}**	**X^{14}**	**X^{15}**
1	1	**1**	1	**1**	1	**1**	1	**1**	1	**1**	1	**1**	1	**1**
2	4	**8**	1	**2**	4	**8**	1	**2**	4	**8**	1	**2**	4	**8**
3	9	**12**	6	**3**	9	**12**	6	**3**	9	**12**	6	**3**	9	**12**
4	1	**4**	1	**4**	1	**4**	1	**4**	1	**4**	1	**4**	1	**4**
5	10	**5**	10	**5**	10	**5**	10	**5**	10	**5**	10	**5**	10	**5**
6	6	**6**	6	**6**	6	**6**	6	**6**	6	**6**	6	**6**	6	**6**
7	4	**13**	1	**7**	4	**13**	1	**7**	4	**13**	1	**7**	4	**13**
8	4	**2**	1	**8**	4	**2**	1	**8**	4	**2**	1	**8**	4	**2**
9	6	**9**	6	**9**	6	**9**	6	**9**	6	**9**	6	**9**	6	**9**
10	10	**10**	10	**10**	10	**10**	10	**10**	10	**10**	10	**10**	10	**10**
11	1	**11**	1	**11**	1	**11**	1	**11**	1	**11**	1	**11**	1	**11**
12	9	**3**	6	**12**	9	**3**	6	**12**	9	**3**	6	**12**	9	**3**
13	4	**7**	1	**13**	4	**7**	1	**13**	4	**7**	1	**13**	4	**7**
14	1	**14**	1	**14**	1	**14**	1	**14**	1	**14**	1	**14**	1	**14**

Tabela 3 - Exponenciação módulo 15
Table 3 - Exponentiation modulo 15

Como pode ser observado, esta tabela apresenta um padrão repetitivo que ocorre com uma periodicidade diferente daquela vista acima. Veremos que esta periodicidade corresponde a um parâmetro conhecido como função Totiente de *N*, denotada por $\varphi(N)$.

Esta função é dada pela contagem dos números no conjunto Z_N que sejam relativamente primos de N. Ou seja, números menores que N e que não possuam fatores comuns a N. Assim, por exemplo, temos: $\varphi(15) = 8$, uma vez que o conjunto de números relativamente primos de 15 é formado por 8 elementos (1, 2, 4, 7, 8, 11, 13, 14). Da mesma forma, $\varphi(11) = 10$, uma vez que o conjunto de números relativamente primos de 11 é formado por 10 elementos (1, 2, 3, 4, 5, 6, 7, 8, 9, 10).

Vemos então que $\varphi(N) = N-1$ para o caso de N primo, pois um número primo é relativamente primo a todos os demais números não nulos em Z_N. De fato, já havíamos observado que era esta a periodicidade com que se repetiam os números elevados a expoentes sucessivos na álgebra módulo N para o caso de N primo. Para o caso de $N = p\ q$, onde p e q são primos, temos o seguinte teorema:

Teorema 2:

Seja $N = p\ q$, onde p e q são ambos números primos. Então:

$$\varphi(N) = (p-1)\ (q-1) \tag{6}$$

Para provar este teorema, podemos tomar o conjunto de todos os números menores ou iguais a $N = p\ q$ e subtrair os números que possuem um fator comum a N. Estes números são $(p, 2p, 3p, \ldots, (q-1)p, q, 2q, 3q, \ldots, (p-1)q, pq)$. Ao todo eles são em número de $p + q - 1$. Dessa forma, $\varphi(N) = pq - p - q + 1$, ou ainda, $\varphi(N) = (p-1)\ (q-1)$.

Podemos agora mostrar que as colunas da Tabela 3 se repetem com uma periodicidade igual a $\varphi(N)$, o que é equivalente a dizer que $X^{\varphi(N)+1} \bmod N = X$.

Teorema 3

Seja $Z_N = \{0, 1, 2, \ldots N-1\}$ e seja $\varphi(N)$ a função totiente de N, então:

$$X^{\varphi(N)+1} \bmod N = X. \tag{7}$$

A demonstração deste teorema exige a definição do subconjunto

$$Z_N^* = \{X \in Z_N \mid \text{existe } X^{-1} \bmod N\}$$

Notemos que a existência de X^{-1} implica $Y \in Z_N$ tal que $(XY) \bmod N = 1$, ou

equivalentemente, que X seja não relativamente primo de N. Pela definição da função totiente, este conjunto deve ter $\varphi(N)$ elementos. Esses elementos são exatamente aqueles que restaram na contagem realizada na demonstração do Teorema 2. Tomemos agora a, um elemento qualquer de Z_N^* e multipliquemos cada um dos elementos de Z_N^* por a. Teremos então o conjunto:

$$a\ Z_N^* = \{ab \bmod N,\ b \in Z_N^*\}$$

Uma vez que a é não relativamente primo de N, temos que cada elemento de $a\ Z_N^*$ é também elemento de Z_N^*. Além disso, como existe a^{-1}, temos que $ab \bmod N = ac \bmod N$ implica $b = c$. Dessa forma:

$$a\ Z_N^* = Z_N^*$$

Ou seja, os elementos que compõem o primeiro conjunto também compõem o segundo e vice-versa. Multiplicando todos os elementos de Z_N^* e, da mesma forma, todos os elementos de $a\ Z_N^*$, temos, considerando a igualdades dos resultados, após uma breve simplificação:

$$a^{\varphi(N)} \bmod N = 1$$

Este resultado vale para qualquer elemento de Z_N que possua um elemento inverso. De onde podemos escrever:

$$X^{\varphi(N)+1} \bmod N = X.$$

Vamos mostrar que o mesmo resultado se aplica para os demais elementos, os quais não possuem inverso. Estes elementos pertencem a dois subconjuntos distintos: um deles formado pelos múltiplos de p e outro formado pelos múltiplos de q. Ou seja, $\{p, 2p, \ldots (q-1)p\}$ e $\{q, 2q, \ldots (p-1)q\}$. É notável que as operações de multiplicação módulo N entre elementos destes subconjuntos produzem resultados também contidos nestes subconjuntos. Dessa forma, as operações de exponenciação são fechadas para estes subconjuntos, os quais também possuem outras propriedades próprias dos campos produzidos por números primos (semelhantes aos da Tabela 1). Notemos que, inclusive, o número de elementos coincide com aqueles gerados por números primos. Apesar de não possuírem explicitamente o número 1, outro elemento faz a vez do elemento neutro

multiplicativo nestes subconjuntos. Notemos, na Tabela 3, a existência do número 6 e do número 10, que representam os elementos neutros multiplicativos, respectivamente, para o subconjunto de múltiplos de 3 e de múltiplos de 5 para a operação de multiplicação módulo 15.

Uma vez que os subconjuntos formados pelos múltiplos de *p* representam campos semelhantes aos gerados por números primos para a exponenciação módulo *N*, temos que, para *X* múltiplo de *p*, vale a relação: $X^p \ mod \ N = X$. Além disso, pelo comportamento repetitivo da função exponencial, temos que

$$X^{k(p-1)+1} \ mod \ N = X \qquad para \ X = jp$$

Analogamente, para os múltiplos de q, temos

$$X^{k(q-1)+1} \ mod \ N = X \qquad para \ X = jq$$

Usando (6), estas duas equações nos permite concluir que também para os elementos relativamente primos de *N* vale a relação (7), ficando assim provado o Teorema.

O ALGORITMO RSA

O Algoritmo RSA é implementado a partir da escolha de dois números primos grandes, *p* e *q*. Para assegurar a inviolabilidade do sistema, estes números devem possuir em torno de uma centena de casas decimais [SIMANCA & SUTHERLAND]. Multiplicando estes números, é obtido *N*, o módulo das operações.

Tal qual vimos acima, as chaves pública e privada são implementadas a partir das expressões:

$$C = M^e \ mod \ N$$

e

$$M = C^d \ mod \ N$$

onde *M* constitui a informação a ser codificada, *C* é o resultado do processo de cifragem e os parâmetros *d* e *e* são agora escolhidos de forma similar à equação (5). Porém, com

a seguinte generalização:

$$[e\ d]\ mod\ \varphi(N) = 1 \tag{8}$$

onde $\varphi(N)$ é facilmente obtido através da equação (6) presumindo-se que sejam conhecidos *p* e *q*, os fatores de *N*. Além disso, *d* e *e* devem ser relativamente primos de $\varphi(N)$. De outra forma, por exemplo, $\varphi(N)$ teria um fator comum com *e*, e o produto *[e d]* teria também este fator, de forma que a equação (8) não poderia ser atendida. Uma escolha deste tipo para *e* produziria uma codificação conforme uma das colunas que não estão em negrito na Tabela 3, onde valores diferentes de *M* podem gerar um mesmo *C*.

De uma forma cômoda, o valor de *e* pode ser escolhido como sendo um número primo, que estará atendida a condição de ser relativamente primo de $\varphi(N)$. Já a obtenção de *d* é um pouco artificiosa. A forma utilizada para obter este número faz uso de um conhecido método para encontrar o mdc (máximo divisor comum) entre dois números, também conhecido como Algoritmo Euclidiano.

O exemplo a seguir ilustra a utilização do Algoritmo Euclidiano para identificar o mdc entre os números 136 e 104. Pode-se observar que o mdc é dado pelo número 8, que figura no canto superior direito, resultante de uma divisão exata.

136	104	32	8
32	1	3	4

Como exemplo da utilização deste método para definir os parâmetros do Algoritmo RSA, suponhamos $\varphi(N)$ = 3220 e *d* = 53, onde o valor de *d* foi definido como um número primo arbitrário. A aplicação do Algoritmo Euclidiano permite obter o seguinte resultado:

3220	53	40	13	1
40	60	1	3	13

O número 1 no canto superior direito da tabela indica que $\varphi(N)$ **e** *e* possuem mdc = 1 ou, de forma equivalente, que são relativamente primos. Além disso, a seqüência de

números obtidos pelo Algoritmo Euclidiano permite escrever o seguinte conjunto de equações:

$$\begin{aligned} \varphi(N) &= 3220 \\ e &= 53 \\ \varphi(N) - 60\,e &= 40 \\ -\varphi(N) + 61\,e &= 13 \\ 4\varphi(N) - 243\,e &= 1 \end{aligned}$$

Aplicando a função módulo na última equação, aplicando (8) e notando que (-243) mod 3220 = 2977, temos:

$$d = 2977$$

O valor de *d*, juntamente com *N*, o módulo das operações, é utilizado na composição da chave privada, enquanto que *e* e *N* são distribuídos na forma da chave pública. As equações (1) e (2) representam as operações realizadas por estas chaves, que permitem criptografar e decriptografar as informações, perfazendo funções inversas.

É importante que, embora matematicamente relacionadas, as chaves pública e privada, tenham características que impeçam que o conhecimento de uma delas possibilite o descobrimento da outra.

A rigor, é sempre possível obter a outra chave quando se conhece uma delas. O que precisa ser considerado é o esforço matemático exigido para este fim. De fato, conhecendo-se a chave pública (*N* e *e*) é sempre possível encontrar $\varphi(N) = (p-1)(q-1)$ a partir da fatoração de *N*, e daí encontrar *d* pelo mesmo método originalmente usado, o Algoritmo Euclidiano. Entretanto, a fatoração de números grandes é um problema de difícil solução. Embora existam métodos sofisticados, que otimizam o processo de busca, todos eles se utilizam de técnicas numéricas e baseiam-se em critérios de tentativa e erro. A ordem de grandeza dos números utilizados na geração das chaves do Algoritmo RSA é o único empecilho para a quebra de sua segurança. Os números utilizados normalmente apontam para um esforço computacional equivalente a séculos ou milênios em uma máquina de última geração. Quando o grau de segurança oferecido por estes números for insuficiente, é apenas uma questão de acrescentar mais algumas casas decimais aos parâmetros do algoritmo [RAITER].

Consideremos que um número de 200 casas decimais, quando convertido para binário

apresenta cerca de 664 bits, ou ainda 111 caracteres (de 6 bits cada), e ocupam igual número de bytes em um arquivo de texto e, lembrando que cada chave possui pelo menos dois números com essa característica (*N* e *e*), temos que a chave pública do Algoritmo RSA deve ter sua distribuição realizada através de um arquivo de dados.

IMPLEMENTAÇÃO DO ALGORITMO RSA

É um fato conhecido que a criptografia de chave pública demanda um grande esforço computacional. As operações com números grandes exigem cuidados especiais para evitar que as operações produzam números imensos, que estourem a capacidade de memória do computador.

A título de ilustração, suponhamos *M* com 100 casas decimais e *e* com 50 casas decimais. Dessa forma, para o cálculo da chave pública do Algoritmo RSA, teremos que calcular a expressão (1), onde:

$$M^e \approx [10^{100}]10^{50} = 10^{10^{52}}$$

Ou seja, M^e apresenta aproximadamente 10^{52} casas decimais. É importante observar que um número desta grandeza não poderia ser armazenado em lugar algum, nem mesmo somando-se todos os recursos de memória existentes no planeta.

No entanto, é possível obter o módulo deste número, evitando-se a explosão de memória, através de alguns artifícios matemáticos. Vamos demonstrar este método através de um exemplo, calculando o valor de $[2221]^{133}$ mod 5000. Para isso, iniciaremos obtendo recursivamente esta mesma base elevada a todas as potências de dois, conforme abaixo:

$[2221]^1 \bmod 5000 = 2221$
$[2221]^2 \bmod 5000 = 4932841 \bmod 5000 = 2841$
$[2221]^4 \bmod 5000 = [2841]^2 \bmod 5000 = 1281$
$[2221]^8 \bmod 5000 = [1281]^2 \bmod 5000 = 961$
$[2221]^{16} \bmod 5000 = [961]^2 \bmod 5000 = 3521$
$[2221]^{32} \bmod 5000 = [3521]^2 \bmod 5000 = 2441$
$[2221]^{64} \bmod 5000 = [2441]^2 \bmod 5000 = 3481$
$[2221]^{128} \bmod 5000 = [3481]^2 \bmod 5000 = 2361$

Decompondo o expoente 133 em potências de 2, temos:

$$133 = 128 + 4 + 1$$

Dessa forma:

$$[2221]^{133} = [2221]^{128} \times [2221]^{4} \times [2221]^{1}$$

$$[2221]^{133} \bmod 5000 = [2361 \times 1281 \times 2221] \bmod 5000$$
$$= 3024441 \bmod 5000 \times 2221 \bmod 5000$$
$$= [4441 \times 2221] \bmod 5000$$
$$= 9863461 \bmod 5000 = 3461$$

Assim, as operações de exponenciação com módulo são reduzidas a uma tabela de multiplicações, onde o número de casas decimais armazenadas permanece dentro de uma faixa exeqüível.

Outro problema que surge na implementação do Algoritmo RSA é a necessidade de gerar números primos aleatórios com um grande número de casas decimais. Para contornar este problema, a abordagem mais prática é gerar primeiramente o número e, a seguir, testá-lo para verificar se ele é primo ou não.

O teste é relativamente simples, porém, dada a ordem de grandeza dos números, sua realização sempre acaba representando um considerável esforço computacional. Dessa forma, é desejável que os números sejam preliminarmente analisados, para evitar o consumo de processamento com candidatos que não possuem chance de serem primos. Por exemplo, os números pares ou terminados em 5, não devem ser testados, pois são múltiplos de 2 e/ou de 5. Além disso, é possível saber se um número é múltiplo de 3 pela simples soma dos seus algarismos (que, no caso resulta em um número divisível por três). Esta operação deve ser realizada efetivamente, uma vez que reduz consideravelmente as chances de insucessos.

Uma regra para a realização dos testes para a geração dos números primos pode ser como a apresentada a seguir:

- Gerar um número grande aleatoriamente
- Eliminar o número caso seja um mau candidato
- Testar para verificar se o número é primo
- Repetir todo o processo no caso de insucesso

Um teste simples e que oferece uma grande certeza sobre a primalidade de um número é dado pelo teorema de Fermat, apresentado acima como Teorema 1:$X^{N-1} \bmod N = 1$

Ou seja, se N é um número primo, então o módulo N de qualquer elemento de Z_N elevado a $N-1$ deverá resultar em 1. É claro que para um teste eficaz vários valores devem ser testados para X e em todos o mesmo resultado deve ser obtido. No caso de um resultado diferente do esperado, o número deve ser descartado, uma vez que não se trata de um número primo. Ainda que este procedimento ofereça um elevado grau de certeza sobre a primalidade de um número, ele não constitui uma prova absoluta de que um número seja primo [CALDWELL]. Ele somente oferece plena certeza em relação aos números que não passam no teste e devem ser rejeitados. Ainda que seja uma ocorrência muito rara, são conhecidos números não primos que, mesmo assim, atendem a relação definida pelo Teorema de Fermat. Para que o processo de escolha não seja comprometido, outros testes mais eficazes e mais complexos devem ser realizados.

Embora a geração de números primos com muitas casas decimais seja um processo computacionalmente dispendioso, esta rotina é utilizada somente na geração do par de chaves e não no processamento criptográfico envolvido no envio e recebimento de mensagens. Ainda assim, é fato conhecido que a criptografia de chave assimétrica exige muito esforço computacional. Esta característica é resultante da série de operações que precisam ser realizadas para evitar a explosão de memória, além do próprio uso de números grandes.

De certa forma, a complexidade do algoritmo é um fator interessante, pois garante sua inviolabilidade. Enquanto o aumento do número de casas decimais acrescenta uma complexidade aproximadamente proporcional ao número de casas adicionadas, o que impõe um aumento linear no tempo requerido para a codificação, o esforço exigido para a quebra do algoritmo aumenta de forma vertiginosa podendo representar milênios.

Normalmente a criptografia de chave assimétrica é usada para transmitir pequenas quantidades de informação, especialmente quando ao processo de criptografia está associado um algoritmo de chave simétrica. Neste caso, a criptografia de chave assimétrica é usada unicamente para enviar a “chave secreta” que será usada no outro processo criptográfico, de menor complexidade numérica. Dessa forma, a criptografia de chave pública constitui uma importante solução para o problema de distribuição de chaves que, de outra forma, teria que ser feita através de um meio alternativo confiável,

o que nem sempre se encontra disponível. Além do que, havendo a disponibilidade desse meio, é bem provável que se possa prescindir da criptografia.

CONCLUSÃO

O Algoritmo RSA constitui um exemplo de aplicação de várias teorias matemáticas em uma solução bastante elegante para o problema de criptografia assimétrica, ou de chave pública, onde as partes não possuem uma chave secreta previamente definida e dependem de um canal inseguro para se comunicar, como é o caso da Internet. Dessa forma, esse algoritmo se aplica perfeitamente em transações eletrônicas envolvendo negócios e/ou comercio pela Internet.

Ainda que apresente uma estrutura elementar, o Algoritmo RSA utiliza números com muitas casas decimais, o que exige cuidados especiais para evitar a explosão da capacidade de memória das máquinas.

A estrutura algébrica do algoritmo foi discutida sob uma abordagem didática, em que o comportamento periódico dos números pode ser observado através de uma tabela e suas colunas.

REFERÊNCIAS BIBLIOGRÁFICAS

CALDWELL C., "Finding primes & proving primality", The Prime Pages, http://www.utm.edu/research/primes/

FLINN, P. J. & JORDAN, J. M. III, "Using the RSA Algorithm for Encryption and Digital Signatures: Can You Encrypt, Decrypt, Sign and Verify without Infringing the RSA Patent?", Cyberlaw http://www.cyberlaw.com/rsa.html, July, 1997

RAITER, B., "Prime Number Hide-and-Seek: How the RSA Cipher Works", Muppetlabs, http://www.muppetlabs.com/~breadbox/txt/rsa.html

RIVEST, R. L., SHAMIR, A., ADLEMAN, L., "On Digital Signatures and Public Key Cryptosystems," MIT - Laboratory for Computer Science Technical Memorandum 82, April, 1977.

SCHNEIER, B., Applied Cryptography: Protocols, Algorithms, and Source in C. 2nd Ed., John Wiley & Sons, Inc: New York. 1996.

SIMANCA S. R. & SUTHERLAND, S.,"Mathematical Problem Solving with Computers", Summer 2002, http://mathlab.mathlab.sunysb.edu/~scott/Book331/