V Bienal da SBM
Sociedade Brasileira de Matemática
UFPB - Universidade Federal da Paraíba
18 a 22 de outubro de 2010

ALGORITMOS DE PRIMALIDADE NA CRIPTOGRAFIA RSA

JOSÉ SÉRGIO DOMINGUES *

Resumo

Apresentaremos nesse trabalho os princípios básicos do método de criptografia RSA e também analisaremos três algoritmos de verificação de primalidade, que podem ser utilizados em programas baseados nesse método e que não requerem alto nível de segurança.

Fundamentação Teórica da Criptografia RSA:

As mensagens transmitidas pela internet podem ser interceptadas com certa facilidade, e isso torna necessário codificar essas mensagens. Por exemplo, certas transações bancárias ou comerciais contêm informações sigilosas que não podem cair nas mãos de um hacker. Para manter sigilosas tais informações foi necessário inventar códigos difíceis de serem quebrados, mesmo com a ajuda de computadores. Os criptossistemas atuais são de chave pública. O nome chave pública se deve ao fato da função codificadora ser de conhecimento geral, mesmo assim um usuário não-legítimo não consegue decodificar a mensagem usando a mesma função para determinar a função decodificadora. Isto significa que o conhecimento da função usada para codificar a mensagem não implica que a mesma será usada para decodificá-la. Cada usuário do sistema de chave pública divulga sua chave codificadora, permitindo assim que qualquer um envie mensagens, mas somente o usuário que conhece a chave decoficadora é capaz de ler as mensagens. Concluímos então que em um código de chave pública, saber codificar não implica em saber decodificar. O mais conhecido destes métodos é o R.S.A., criado em 1978, por R. L. Rivest, A. Shamir e L. Adleman, quando trabalhavam no Massachussets Institute of Technology (M.I.T). Para entendê-lo melhor é necessário saber alguns conceitos de Teoria dos Números, como números primos, congruência modular e unidades em $\mathbb{Z}_n$. Esses conceitos serão rapidamente relembrados abaixo:

Um número inteiro positivo $p \neq 1$ é dito **primo**, se possui apenas dois divisores.

Dizemos que dois inteiros a e b são **congruentes módulo** n se, e somente se, a e b deixam o mesmo resto na divisão por n. Quando isso ocorrer, escrevemos

$$a \equiv b \ (mod\ n)$$

Definimos o anel de inteiros módulo n, $\mathbb{Z}_n$, como o conjunto de classes módulo n. Assim

$$\mathbb{Z}_n = \left\{\overline{0},\ \overline{1},\ ...,\ \overline{n-1}\right\}$$

Exemplos: $\mathbb{Z}_4 = \left\{\overline{0},\ \overline{1},\ \overline{2},\ \overline{3}\right\}$ e $\mathbb{Z}_6 = \left\{\overline{0},\ \overline{1},\ \overline{2},\ \overline{3},\ \overline{4},\ \overline{5}\right\}$

Dizemos que a é uma **unidade** em $\mathbb{Z}_n$ quando $a \cdot x = 1$, para algum $x \in \mathbb{Z}_n$.

*Mestrando em Modelagem Matemática e Computacional do CEFET-MG e Professor do Departamento de Ciências Exatas da Universidade Estadual de Montes Claros - UNIMONTES e da Faculdade de Ciências e Tecnologia de Montes Claros - FACIT, Brasil, jsergio_ mat@lsi.cefetmg.br.

Se p e q são dois números primos, definimos a função φ de Euler da seguinte forma:

$$\varphi(p) = p - 1 \text{ e } \varphi(pq) = (p-1)(q-1)$$

A segurança desse método está no fato de ser extremamente difícil, mesmo que computacionalmente, fatorar números muito grandes. Como já foi dito, a implementação do RSA se deve à chamada chave pública, que é o par $(n,\ \mathbf{e})$, onde n é o produto de dois números primos muito grandes e $\mathbf{e}$ é um inteiro entre 1 e $\varphi(n)$, tal que $mdc\ (\varphi(n),\ \mathbf{e}) = 1$. Com isso, se consegue codificar uma determinada mensagem, porém, como a chave é pública, qualquer pessoa que a encontre pode codificar uma mensagem, mas para decodificá-la, é preciso possuir a chave privada, que é o par $(n,\ d)$, onde d é um inteiro tal que, $d \cdot \mathbf{e} = 1\ (mod\ \varphi(n))$, ver Coutinho [5].

A segurança desse método está no fato de ser extremamente difícil, mesmo que computacionalmente, fatorar números muito grandes. Como já foi dito, a implementação do RSA se deve à chamada chave pública, que é o par $(n,\ \mathbf{e})$, onde n é o produto de dois números primos muito grandes e $\mathbf{e}$ é um inteiro entre 1 e $\varphi(n)$, tal que $mdc\ (\varphi(n),\ \mathbf{e}) = 1$. Com isso, se consegue codificar uma determinada mensagem, porém, como a chave é pública, qualquer pessoa que a encontre pode codificar uma mensagem, mas para decodificá-la, é preciso possuir a chave privada, que é o par $(n,\ d)$, onde d é um inteiro tal que, $d \cdot \mathbf{e} = 1\ (mod\ \varphi(n))$, ver Coutinho [5].

Metodologia:

Descreveremos abaixo como se proceder matematicamente para codificar e também decodificar uma determinada mensagem pelo método em questão.

1. Escolhe-se p e q primos.
2. Calcula-se $n = pq$.
3. Calcula-se $\varphi(n) = (p-1)(q-1)$.
4. Encontra-se $\mathbf{e}$, tal que $1 < \mathbf{e} < \varphi(n)$ com $mdc(\varphi(n),\ \mathbf{e}) = 1$.
5. Encontra-se d, tal que $d \cdot \mathbf{e} \equiv 1\ (mod\ \varphi(n))$, isso é, d é o inverso de $\mathbf{e}$ $mod\ (\varphi(n))$.
6. O par $(n,\ e)$ será a chave pública.
7. O par $(n,\ d)$ será a chave privada.
8. Nesse trabalho, usaremos a tabela ASCII para converter cada letra da mensagem em números. Feito isso, teremos uma seqüência de números que deve ser quebrada em blocos de valores menores do que n. Cada bloco m será codificado da seguinte forma:
$$C(m) = m^e\ (mod\ n) = k,$$
isso é, $C(m)$, que é a codificação do bloco m é o resto da divisão de m^e por n.
9. Para recuperar a mensagem codificada basta fazer outra potenciação modular, onde k é um bloco de mensagem já codificada. Veja:
$$D(k) = k^d\ (mod\ n)$$

Como exemplo, vamos codificar a palavra MESTRADO, utilizando, para fins meramente pedagógicos, $p = 7$ e $q = 11$, números primos bem pequenos.

- De acordo com os valores de p e q, temos que $n = 7 \cdot 11 = 77$ e $\varphi(n) = (7-1)(11-1) = 60$.
- $60 = 2^2 \cdot 3 \cdot 5$, podemos tomar $\mathbf{e} = 7$, já que $1 < 7 < 60$ e $mdc(60,\ 7) = 1$, e essa última condição, garante que $\mathbf{e}$ é uma unidade em $mod(\varphi(n)) = mod(60)$.
- Concluímos então, que a chave pública (chave de codificação) é o par $(77,\ 7)$
- Vamos agora, calcular d, que é o inverso de $\mathbf{e}\ \ mod(\varphi(n))$. Queremos encontrar uma solução para a congruência $d \cdot 7 \equiv 1\ (mod\ 60)$. Observe que $d = 43$ é uma solução, pois $7 \cdot 43 = 301 \equiv 1\ (mod\ 60)$.
- Logo, a chave privada (chave de decodificação) é o par $(77,\ 43)$.
- As numerações respectivas das letras de MESTRADO na tabela ASCII são 77, 69, 83, 84, 82, 65, 68 e 79. Desta forma, a sequência numérica a ser usada é 7769838482656879, que deve ser quebrada em blocos de valores menores do que 77. Tomemos os blocos 7, 7, 69, 8, 38, 48, 26, 56, 8, 7, 9. A codificação de cada bloco acima é dada por:

$$C(7) = 7^7(mod\ 77) = 28 \qquad C(69) = 69^7(mod\ 77) = 20$$

$$C(8) = 8^7(mod\ 77) = 57 \qquad C(38) = 38^7(mod\ 77) = 3$$

$$C(48) = 48^7(mod\ 77) = 27 \qquad C(26) = 26^7(mod\ 77) = 5$$

$$C(56) = 56^7(mod\ 77) = 56 \qquad C(9) = 9^7(mod\ 77) = 37$$

Portanto, a mensagem enviada, já criptografada será

$$28 - 28 - 20 - 57 - 3 - 27 - 5 - 56 - 57 - 28 - 37$$

Para decodificar essa mensagem, basta lembrar que $D(k) = k^d\ (mod\ n)$. Logo

$$D(28) = 28^{43}(mod\ 77) = 7 \qquad D(20) = 20^{43}(mod\ 77) = 69$$

$$D(57) = 57^{43}(mod\ 77) = 8 \qquad D(3) = 3^{43}(mod\ 77) = 38$$

$$D(27) = 27^{43}(mod\ 77) = 48 \qquad D(5) = 5^{43}(mod\ 77) = 26$$

$$D(56) = 56^{43}(mod\ 77) = 56 \qquad D(37) = 37^{43}(mod\ 77) = 9$$

Dessa forma, a sequência decodificada será

$$77 - 69 - 83 - 84 - 82 - 65 - 68 - 79$$

que corresponde, via tabela de conversão, à palavra MESTRADO.

Apresentando e Discutindo os Algoritmos:

Na figura 1, vamos conhecer dois dos três algoritmos de verificação de primalidade mencionados na introdução, o primeiro verifica se um dado número inteiro positivo p é primo, através de divisões sucessivas de p por todos os inteiros de 1 até p, como descrito em Cavalcante [3]. Já o segundo algoritmo, foi uma melhoria que fizemos, utilizando um resultado muito interessante e de demonstração relativamente simples da teoria dos números, que nos garante que: *para determinar se um dado número inteiro positivo n é primo, basta verificar se o mesmo não é divisível por nenhum primo menor do que* $\sqrt{n}$, Hefez [7].

Observe que a modificação que fizemos gera um algoritmo que diminui consideravelmente o número de divisões executadas para efetuar a verificação desejada.

```
int primo ( int p )
{
   int i, pri = 0, div = 0;
   for ( i = 1; i <= p; i++ )
      if ( p%i = = 0 ) div++;
   if ( div = = 2 ) pri = 1;
   return ( pri );
}
```

(a)

```
int primo ( int n )
{
   int raiz = (int) sqrt(n), cont = 0;
   for ( int i = 2; i <= raiz; i++ )
   {
      if ( n%i = = 0 ) cont++;
      if ( cont = = 0 ) return = 1;
      else
          return 0;
     }
}
```

(b)

Figura 1: Algoritmos de verificação de primalidade, onde o primeiro executa a divisão de p por cada um dos inteiros de 1 até p, ou seja, p divisões. O segundo, efetua apenas as divisões de p por todos os inteiros entre 1 e $\sqrt{p}$, o que diminui drasticamente o número de divisões efetuadas. *A figura 1-(a) foi adaptada de Mole [9].*

Percebemos facilmente que os algoritmos (a) e (b) apresentados na figura 1 tem complexidades de tempo respectivamente iguais a

$$\boxed{T(n) = O(n)} \quad \text{e} \quad \boxed{T(n) = O(n^{1/2})},$$

mas, considerando o tamanho das entradas como números binários, temos que o seu tamanho não é n, e sim, $log\ n$. Ver mais em Castonguay [2] e Filho [6]. Com isso, podemos verificar de maneira simples, que as complexidades dos algoritmos da figura 1 são, na realidade, exponenciais, o que os torna inviáveis para testar a primalidade de números muito grandes.

O terceiro algoritmo que apresentaremos é o famoso *Crivo de Eratóstenes*. Esse algoritmo é um clássico da teoria dos números e tem a finalidade de encontrar todos os números primos menores do que ou iguais a um número inteiro positivo n previamente estabelecido. Ele começa com a lista de números inteiros positivos de 2 até n. A cada passo, o menor fator primo, p, que ainda não foi utilizado é escolhido e todos os múltiplos desse fator são removidos da lista. O algoritmo termina quando todos os fatores primos menores do que, ou iguais a $\sqrt{n}$, já tiverem

sido utilizados, Albuquerque [1] e Júnior [8].

Considerando $n = 20$, temos que $\sqrt{n} = \sqrt{20} \approx 4,47$ e, consequentemente, vamos considerar $\lfloor \sqrt{n} \rfloor = \lfloor \sqrt{20} \rfloor = 4$. É óbvio que os números primos menores do que 4 são 2 e 3, ou seja, na lista de inteiros que vai de 2 a 20, devemos remover todos os múltiplos de 2 (maiores do que 2) e logo em seguida, todos os múltiplos de 3 (maiores do que 3). O crivo nos garante que, ao final desse processo, os números que não foram removidos formam o conjunto de todos os primos menores do que 20.
Na figura 2, abaixo, podemos ver com clareza uma exemplificação desse algoritmo, para o caso onde $n = 20$. Nela, os números que estão sombreados representam os múltiplos de 2 ou 3 e os que não estão sombreados formam o conjunto

$$\mathcal{P}_{20} = \{2,\ 3,\ 5,\ 7,\ 11,\ 13,\ 17,\ 19\}$$

de todos os números primos menores do que 20.

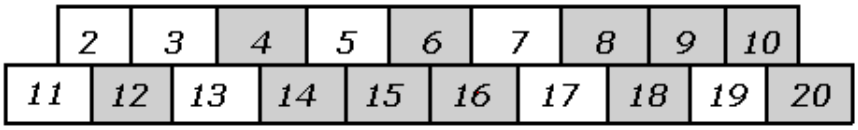

Figura 2: Exemplificação da utilização do Crivo de Eratóstenes quando n = 20 e, portanto, deve-se marcar todos os múltiplos de 2 e 3.

Segue abaixo, o pseudocódigo do Crivo de Eratóstenes, que foi encontrado em Júnior [8].

```
{Inicializa a lista}
for i ← 2 até n do
  Lista_i ← Desmarcado
end for
p ← 2
while p^2 ≤ n do
  {Marca os múltiplos de p}
  k ← p^2
  while k ≤ n do
    Lista_k ← Marcado
    k ← k + p
  end while
  {Escolhe o próximo fator primo p}
  while Lista_p ≠ Desmarcado do
    p ← p + 1
  end while
end while
{Imprime a lista}
for i ← 2 até n do
  if Lista_i = Desmarcado then
    print i
  end if
end for
```

Figura 3: Pseudocódigo do Crivo de Eratóstenes.

Analisando a complexidade de tempo de cada um dos três laços do algoritmo da figura 3, é possível mostrar que a complexidade de tempo, T(n), do Crivo de Eratóstenes no pior caso, melhor caso e caso médio é

$$\boxed{T(n) = O(n \cdot log\ log\ n)}$$

Outra aplicação:

Outra aplicação importante para os números primos é a construção de tabelas hash, Cormem [4] e Wikipedia [11]. É muito comum escolher um tamanho da tabela que seja primo. Isso é feito para evitar que valores altos utilizados para endereçar a tabela tenham divisores em comum com o tamanho da tabela, o que poderia induzir um alto número de colisões após a operação de módulo. Logo, dependendo da quantidade de registros a serem armazenados em uma tabela, se faz necessária a utilização de números primos relativamente grandes e para isso, deve-se verificar, com precisão, se um dado número a ser usado é primo ou não. Ver mais detalhes em Ziviane [12].

Conclusões:

Através desse trabalho, podemos, antes de tudo, ter uma boa visão matemática de como funciona o método de criptografia RSA, através da apresentação da fundamentação teórica e da metodologia do método e também da codificação e decodificação de uma palavra.

Ainda apresentamos dois exemplos de algoritmos de verificação de primalidade, porém, ambos ineficientes para números relativamente grandes. Mas vale lembrar, que diminuimos o custo computacional do algoritmo da figura 1-(a), que foi encontrado em Mole [9], com base em um teorema importante da Teoria dos Números, que pode ser encontado em Hefez [7], obtendo, então, o algoritmo da figura 1-(b).

Obviamente, mesmo depois da modificação feita, esse algoritmo ainda fica muito longe de ser eficiente, pois como vimos, sua complexidade é exponencial. Também se observa até hoje, que os programadores tem dificuldades de encontrar um algoritmo eficiente de fatoração de números muito grandes, como foi mencionado, da ordem de 1000 algarismos ou mais. Na realidade, existe uma vertente nessa área, que acredita ser impossível encontrar tal algoritmo.

Já o algoritmo da figura 3, que é o pseudocódigo do Crivo de Eratóstenes, representa uma técnica simples e eficiente de encontrar todos os números primos até uma ordem desejada, porém, não é eficiente para ordens muito elevadas. Pode-se observar que esse algoritmo usa muita memória e executa os laços muitas vezes para chegar ao seu objetivo, o que é ruim. Por outro lado, nesse algoritmo não é necessário executar divisões (que diga-se de passagem, são cálculos lentos), e o algoritmo é muito fácil de programar, o que é muito bom. Portanto, não é o algoritmo ideal para verificação de primalidade para sistemas de criptografia que necessitem de uma confiabilidade elevada, já que não conseguirá, em tempo hábil, efetuar essa verificação.

Referências

[1] ALBUQUERQUE, J. - *O Crivo de Erastóstenes.*, DFM - UFRPE, Recife - PE, 2004.

[2] CASTONGUAY, D. - *Projeto e Análise de Algoritmos*, Instituto de Informática - UFG, 2007.

[3] CAVALCANTE, A.L.B. - *Pirâmides de Números Primos Palíndromos*, UPIS Faculdades Integradas - Faculdade de Tecnologia, Departamento de Sistemas de Informação, 2006.

[4] CORMEM, T.H., LEISERSON, C.E.; RIVEST, R.L.; STEIN, C. - *Algoritmos: teoria e prática*, traduzido por Souza, V.D., Rio de Janeiro, Campus - 1ª edição, 2002.

[5] COUTINHO, S.C. - *Números Inteiros e Criptografia RSA*, Instituto Nacional de Matemática Pura e Aplicada - IMPA, Série de Computação e Matemática, 2ª edição.

[6] FILHO, J.G.S. - *Informação, Codificação e Segurança de Dados*, Unb - Faculdade de Técnologia - Departamento de Engenharia Elétrica.

[7] HEFEZ, A. - *Elementos de Aritmética*, Sociedade Brasileira de Matemática - SBM, Textos Universitários, 2005.

[8] JÚNIOR, F.A.F. - *Projeto e Análise de Algoritmos*, UFMG - DCC, 2006.

[9] MOLE, V.L.D.; MARTINS, J.G. - *Números primos, um algoritmo para geração de sequência*, IV Congresso Brasileiro de Computação - CBComp, 2004.

[10] OLIVEIRA, P.E.R.; OLIVEIRA, R.L.G.; EZEQUIEL, P.T. - *RSA*, Universidade Federal de Uberlândia - UFU, 2004.

[11] WIKIPEDIA - *Hash table, Wikipedia, the free encyclopedia*, 2008. [Online; accessed 09-12-2008].

[12] ZIVIANI, N. - *Projeto de Algoritmos com Implementações em Pascal e C*, Pioneira Thomson, 2004.