

RSA: ALGORITMOS DE CHAVE PÚBLICA

PRIMEIRA PUBLICAÇÃO: ABRIL/1998

QUARTA REVISÃO: DEZEMBRO/2004

Teoria e Implementação

Chave Pública

São algoritmos baseados em propriedades matemáticas que possibilitam o processo de criptografia (**encrypt**) a partir de uma chave de conhecimento público (K_p), mas exige uma outra chave (privada) para a descryptografia (**decrypt**) de conhecimento apenas do usuário que a criou (K_U).

Entendendo o RSA

Criado em 1977 e publicado em 1978 por Ron Rivest, Adi Shamir e Len Adleman do MIT, o RSA baseia-se nas propriedades da aritmética modular ($a \equiv r \pmod{b}$) e na função “totient” de Euler ($\phi(n) = |\{k \in \mathbb{Z}_+ \mid 0 < k < n \text{ e } \text{mdc}(k,n) = 1\}|$).

Sejam M , C , e , d e n inteiros não negativos. Consideremos as funções:

$$C = M^e \pmod{n}$$

$$M = C^d \pmod{n}$$

$$M = (M^e \pmod{n})^d \pmod{n}$$

$$M = (M^e)^d \pmod{n}$$

$$M = M^{ed} \pmod{n}$$

A partir das propriedades da aritmética modular, concluímos que

$$M \equiv M^{ed} \pmod{n} \Rightarrow M^{ed} \equiv M \pmod{n}.$$

Consideremos agora o Teorema de Euler

$$m^{k\phi(n)+1} = m^{k(p-1)(q-1)+1} \equiv m \pmod{n}$$

Vamos escolher e , d inteiros, tais que

$$e \times d = k \times \phi(n) + 1 \text{ de maneira a satisfazer } M^{ed} \equiv M \pmod{n}.$$

$$e \times d = k \times \phi(n) + 1 \Rightarrow e \times d \equiv 1 \pmod{\phi(n)} \Rightarrow d \equiv e^{-1} \pmod{\phi(n)}.$$

Esta equação indica que d é o inverso multiplicativo de $e \pmod{n}$ e portanto, para que exista d nestas condições, devemos escolher e , tal que $\text{mdc}(e, \phi(n)) = 1$.

Algoritmo

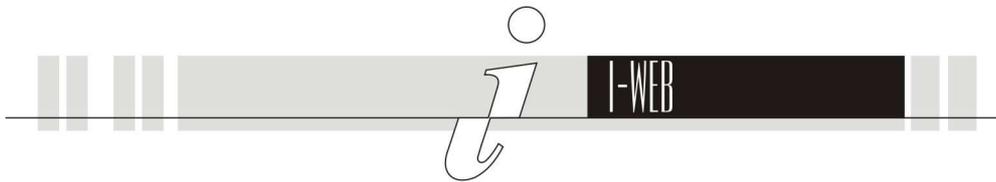
O RSA é estabelecido como:

1. Escolha p e q , 2 inteiros primos (secretos)
2. Calcule $n = p \times q$ (público)
3. Calcule $\phi(n) = (p-1) \times (q-1)$ (segredo)
4. Escolha e inteiro, $1 < e < \phi(n)$ tal que $\text{mdc}(e, \phi(n)) = 1$ (público)
5. Calcule d inteiro, $1 < d < \phi(n)$ tal que $e \times d \equiv 1 \pmod{\phi(n)} \Rightarrow d = (k \times \phi(n) + 1) / e$ (segredo)

As chaves utilizadas serão:

$KP = \{e, n\}$ (pública) e

$KU = \{d, n\}$ (privada)



RSA: ALGORITMOS DE CHAVE PÚBLICA

PRIMEIRA PUBLICAÇÃO: ABRIL/1998

QUARTA REVISÃO: DEZEMBRO/2004

Exemplo

Seguindo os passos do algoritmo:

1. Escolhemos dois primos p e q de acordo com as características de nosso algoritmo. Sejam:

$$p = 1021 = 1111111101_2 \text{ e}$$

$$q = 1019 = 1111111011_2$$

2. Calculamos n , tal que

$$n = p \times q = 1021 \times 1019 = 1040399 \text{ sendo que}$$

$$1040399 = 1111111000000001111_2$$

3. Calculamos $\phi(n)$ tal que

$$\phi(n) = (p-1) \times (q-1) = 1020 \times 1018 = 1038360, \text{ com}$$

$$1038360 = 11111101100000011000_2$$

4. Escolhemos $e = 3577 = 110111111001_2$ de forma a garantir que $\text{mdc}(e, \phi(n)) = 1$

5. Finalmente, calculamos d , satisfazendo

$$1 < d < \phi(n) \text{ e } e \times d \equiv 1 \pmod{\phi(n)}$$

$$d = 426433 = 01101000000111000001_2$$

As chaves de criptografia são:

KP = {3577, 1040399} (chave pública)

KU = {426433, 1040399} (chave privada)

Como n tem 20 bits, dizemos que este é um algoritmo com chave de 20 bits.

A mensagem deve ser transmitida em blocos de tamanho compatível com n , i.e., $0 < M < n$. Portanto, para este exemplo, só podemos transmitir mensagens com até 20 bits.

Aritmética Modular

$b \mid a$

Definição: Sejam a , b dois números inteiros não-negativos.

($0 < a$ e $0 < b$). Dizemos que b "divide" a e representamos por $b \mid a$ se

$a = k \times b$ para algum k inteiro.

$a \equiv r \pmod{b}$

A partir da definição acima, todo número inteiro pode ser escrito na forma

$$a = k \times b + r,$$

onde: b , k e r são inteiros e $0 \leq r < b$ e dizemos que

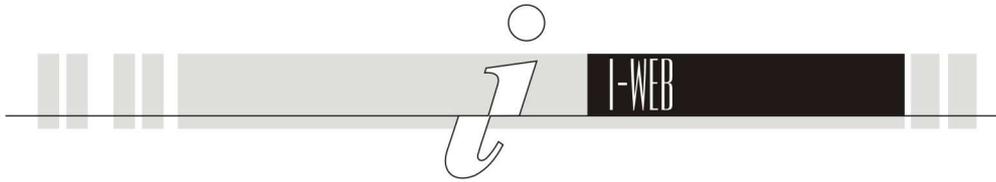
$a \equiv r \pmod{b}$, i.e., $(a-r) = k \times b$ para algum k inteiro.

Propriedade 1:

Uma propriedade interessante pode ser extraída da equação acima:

$$a \equiv r \pmod{b} \Rightarrow (a-r) = k \times b \Rightarrow -1 \times (a-r) = -1 \times k \times b \Rightarrow$$

$$(r-a) = (-1 \times k) \times b \Rightarrow r \equiv a \pmod{b}.$$



RSA: ALGORITMOS DE CHAVE PÚBLICA

PRIMEIRA PUBLICAÇÃO: ABRIL/1998

QUARTA REVISÃO: DEZEMBRO/2004

Função Totient de Euler

Função "totient" de Euler

Consideremos o função de Euler

$$\phi(n) = |\{k \in \mathbb{Z}_+ \mid 0 < k < n \text{ e } \text{mdc}(k,n) = 1\}|,$$

ou em palavras: "o número de inteiros menores do que n que são primos com n".

Propriedade 2:

Seja p um número primo. $\phi(p) = p-1$. É natural, uma vez que como p é primo, $\forall k \in \mathbb{Z}_+, 0 < k < p$, k não divide p (caso contrário p não seria primo).

Propriedade 3:

Sejam p e q dois inteiros primos. Seja $n = p \times q$. Vamos calcular $\phi(n)$.

Como $n = p \times q$, os divisores de n são os números da forma:

$$p, 2 \times p, \dots, (q-1) \times p = \{k \times p \mid 0 < k < q-1\} = P \text{ e}$$

$$q, 2 \times q, \dots, (p-1) \times q = \{k \times q \mid 0 < k < p-1\} = Q.$$

Portanto, $|P| = (q-1)$ e $|Q| = (p-1)$.

$$\begin{aligned} \phi(n) &= \phi(p \times q) = (p \times q - 1) - [|P| + |Q|] = \\ &= (p \times q - 1) - [(q-1) + (p-1)] = p \times q - 1 - q + 1 - p + 1 = \\ &= p \times q - q - p + 1 = (p-1) \times (q-1) = \phi(p) \times \phi(q) \end{aligned}$$

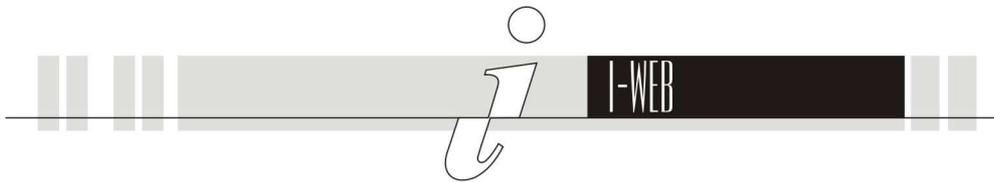
Propriedade 4:

Consideremos o Teorema de Euler: $m^{\phi(n)} \equiv 1 \pmod n$.

O que significa que $(m^{\phi(n)} - 1) = k \times n$ para algum k inteiro.

Alternativamente podemos escrever $m \times (m^{\phi(n)} - 1) = m \times (k \times n) \Rightarrow$

$$m^{\phi(n)+1} - m = (m \times k) \times n \Rightarrow m^{\phi(n)+1} \equiv m \pmod n$$



RSA: ALGORITMOS DE CHAVE PÚBLICA

PRIMEIRA PUBLICAÇÃO: ABRIL/1998

QUARTA REVISÃO: DEZEMBRO/2004

Implementação

Algoritmos

Em geral, os algoritmos para o cálculo dos diversos elementos do sistema RSA são bem simples, entretanto, devido a ordem de grandeza dos números envolvidos nestes cálculos, deve-se tomar cuidados especiais, de maneira a garantir que o modelo utilizado suporte a realização dos cálculos com a precisão necessária.

Obs: Os algoritmos apresentados a seguir têm o objetivo de serem didáticos e não foram otimizados exatamente para ajudar a sua compreensão.

Cálculo dos primos p e q

Seja $p \in \mathbb{Z}_+$

p é primo \Leftrightarrow não existe $k \in \mathbb{Z}_+$ com $1 < k < p$ tal que $k \mid p$.

Note que 1 não é primo, o que está de acordo com a definição.

Podemos restringir a definição de primo ainda mais, pois se $k, x \in \mathbb{Z}_+$ e

$k \mid x$, então $x = k \times j$, portanto $j \mid x$. Seja $n = \min(k, j)$ e $m = \max(k, j)$.

$0 < n \leq \sqrt{x}$, pois caso contrário, como $m \geq n$, $m \times n > \sqrt{x} \times \sqrt{x} > x$.

Então p é primo \Leftrightarrow não existe $k \in \mathbb{Z}_+$ com $1 < k \leq \sqrt{p}$ tal que $k \mid p$.

Seja $Z_p = \{c \in \mathbb{Z}_+ \mid 1 < c < p \text{ e } c \text{ é primo}\}$

Analisando a propriedade da divisão, notamos que:

para $c \in Z_p$ e $k, x \in \mathbb{Z}_+$,

se $c \mid k$ e $k \mid x \Rightarrow k = i \times c$ e $x = k \times j \Rightarrow x = (i \times c) \times j = (i \times j) \times c \Rightarrow c \mid x$
com i e $j \in \mathbb{Z}_+$.

Assim, podemos definir p como primo da forma:

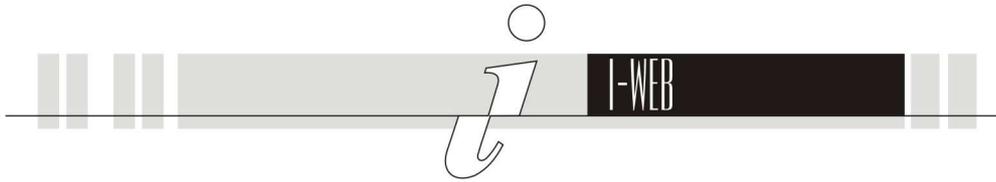
p é primo \Leftrightarrow não existe $k \in Z_p$ com $1 < k \leq \sqrt{p}$ tal que $k \mid p$.

O que significa que basta testarmos os primos menores do que p .

Abaixo apresentamos um algoritmo simples que identifica se p é primo a partir do teste da divisão de p por todo k , $1 < k \leq \sqrt{p}$.

```
#include <math.h>
int pprimo(p)
{
    int k,sqrtp;
    sqrtp = sqrt(p);
    k=2;
    while ((k<=sqrtp) && ((p%k) != 0))
        k++;
    if (k >= sqrtp) return 1;
    else return 0;
}
```

Obs: Note que o algoritmo acima testa $k \in \mathbb{Z}$ com $1 < k \leq \sqrt{p}$, i.e., para todo k e não apenas para todo k primo ($k \in Z_p$). Isto faz o algoritmo mais simples de ser implementado, embora menos eficiente. Na prática, vale a pena manter uma lista com os primeiros 2000 números primos, por exemplo. Porém identificar se um número k é primo é um problema computacional razoavelmente complexo (inclusive é a função deste algoritmo).



RSA: ALGORITMOS DE CHAVE PÚBLICA

PRIMEIRA PUBLICAÇÃO: ABRIL/1998

QUARTA REVISÃO: DEZEMBRO/2004

Escolha de e

Apenas 2 propriedades precisam ser satisfeitas para a escolha de e:

i. $1 < e < \phi(n)$ e

ii. $\text{mdc}(e, \phi(n)) = 1$.

A maneira mais simples de se calcular e, portanto é fatorar $\phi(n)$ e escolher e com fatores diferentes dos fatores de $\phi(n)$. Isto equivale a escolher e entre os primos com $\phi(n)$.

Como fatorar $\phi(n)$ pode não ser muito simples, este método pode não ser muito eficiente, assim podemos, alternativamente, escolher e entre os primos menores que $\phi(n)$. Isto traz a vantagem adicional de facilitar o teste, pois se p é primo, $\text{mdc}(p, \phi(n)) \in \{1, p\}$.

Também podemos notar que se p e q $\neq 2$, então p e q são ímpares e, portanto, $(p-1) \times (q-1)$ é par. Logo $e \neq 2 \times k$ para todo k.

Abaixo descrevemos um algoritmo implementando este caso particular.

```
int eprimofn(phiN)
{
  int e;
  e = 3;
  while ((e < phiN) && (pprimo(e)))
    if ((phiN % e) != 0) return e;
    else e=e+2;
}
```

Cálculo de d

$e \times d \equiv 1 \pmod{\phi(n)}$ e $e \times d = k \times \phi(n) + 1$ para algum $k \in \mathbb{Z}_+$

$d = (k \times \phi(n) + 1) / e$, ou seja, d é o inverso multiplicativo de e mod n.

Na forma como o n foi tomado, dado e, só existe um d que satisfaz esta equação.

Portanto, tomamos um candidato a d e calculamos $r = (d \times e) \pmod{\phi(n)}$.

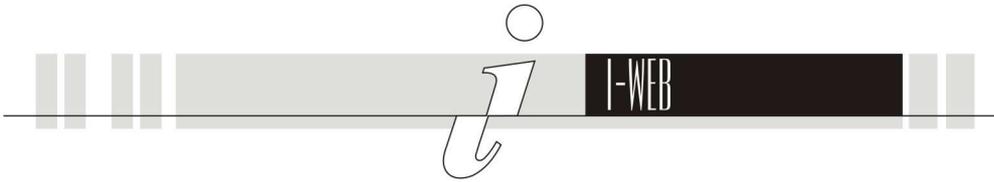
Se $r = 1$, d é válido.

```
int calcd(int phiN)
{
  int d;
  d = 2;
  while ((d < phiN) && (((e*d) % phiN) != 1))
    d++;
  return d;
}
```

Conversão em bits

Um dos primeiros algoritmos que se aprende em computação. A conversão de um número inteiro n da base 10 para a base 2.

```
int bits(int n; int *nbits)
{
  int i;
  i=0;
  while (n > 0) {
    nbits[i] = n % 2;
    n = n / 2;
    i++;
  }
  return (i-1);
}
```



RSA: ALGORITMOS DE CHAVE PÚBLICA

PRIMEIRA PUBLICAÇÃO: ABRIL/1998

QUARTA REVISÃO: DEZEMBRO/2004

Cálculo de $m^e \bmod n$ O processo de cálculo de $C = M^e \bmod n$ envolve a exponenciação de M por e . Há diversas formas de se calcular M^e , como por exemplo:

$M^{16} = M \times M \times \dots \times M$ o que corresponde a 15 multiplicações por M ou ainda podemos calcular:

$M^2 = M \times M$, $M^4 = M^2 \times M^2$, $M^8 = M^4 \times M^4$, $M^{16} = M^8 \times M^8$

o que nos leva ao mesmo resultado em apenas 4 multiplicações.

Além disso, o cálculo de M^e pode, facilmente, ir além da capacidade de manipulação numérica do computador utilizado. Assim, são necessários alguns cuidados no processo de cálculo de M^e .

```
int memodn(int m; int e; int n)
```

```
{
  int *ebits;
  int r, k, i;
  r = 1;
  k = bits(e,ebits);
  for (i=k-1; i>=0; i--)
  {
    r = (r * r) % n;
    if (ebits[i] == 1)
      r = (r * m) % n;
  }
  return r;
}
```

Apesar do procedimento acima, o computador pode não ser capaz de manipular números inteiros muito grandes. Neste caso, deve-se optar pelo uso de tipos "long double" para a manipulação numérica e de suas correspondentes funções matemáticas para sua manipulação.

O algoritmo anterior fica:

```
long double MeModN(long double m; long double e; long double n)
```

```
{
  long double r;
  long int k, i;
  int *ebits;
  r = 1;
  k = bits(e, ebits);
  for (i=k-1; i>=0; i--)
  {
    r = fmod((r * r), n);
    if (ebits[i] == 1)
      r = fmod((r * m), n);
  }
  return r;
}
```

Obs: Para o cálculo de chaves de 128 bits ou maiores, mesmo o tipo "long double" pode não ser suficiente. Para isto, deve-se utilizar implementações numéricas especiais, que permitam manipular números de grandeza apropriada.