

## RSA: PUBLIC-KEY ALGORITHMS

FIRST PUBLISHING: APRIL/1998

FOURTH REVISION: DECEMBER/2004

Fernando Rosendo

[[fernando.rosendo@iweb.com.br](mailto:fernando.rosendo@iweb.com.br)]

i.web Labs - Brazil

### Theory and Implementation

#### Public-Key

Algorithms based on mathematical properties which allow the cryptographic process (**encryption**) using public key ( $K_P$ ), but require a private key ( $K_U$ ) (known by the owner only) to decrypt the message.

#### Understanding the RSA

Created in 1977 and published in 1978 by Ron Rivest, Adi Shamir, and Len Adleman from MIT, RSA algorithm is based on properties of modular arithmetic ( $a \equiv r \pmod{b}$ ) and on Euler "phi" (or "totient") function ( $\phi(n) = |\{k \in \mathbb{Z}_+ \mid 0 < k < n \wedge \gcd(k, n) = 1\}|$ ).

Let  $M$ ,  $C$ ,  $e$ ,  $d$  and  $n$  be non-negative integers. Considering the functions:

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

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

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

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

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

Using the modular arithmetic properties, we can conclude that

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

Considering the Euler's Theorem

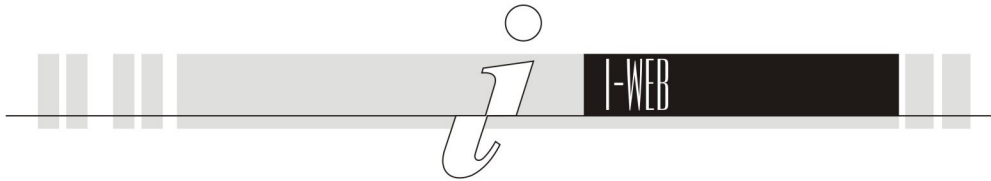
$$m^{kf(n)+1} = m^{k(p-1)(q-1)+1} \equiv m \pmod{n}$$

Let us take  $e$ ,  $d$  integers, such that

$$ed = k\phi(n)+1 \text{ satisfies } M^{ed} \equiv M \pmod{n}.$$

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

This equation shows that  $d$  is the multiplicative inverse of  $e \pmod{n}$ , therefore, we have to choose  $e$ , satisfying  $\text{mdc}(e, \phi(n)) = 1$  to guarantee  $d$  exists under these conditions.



## RSA: PUBLIC-KEY ALGORITHMS

FIRST PUBLISHING: APRIL/1998

FOURTH REVISION: DECEMBER/2004

### Algorithm

The RSA is defined as:

1. Choose **p** and **q**, 2 prime numbers (secret)
2. Calculate **n = p × q** (public)
3. Calculate **φ(n) = (p-1) × (q-1)** (secret)
4. Choose **e** integer, **1 < e < φ(n)** satisfying **mdc(e, φ(n)) = 1** (public)
5. Calculate **d** integer, **1 < d < φ(n)** such that (secret)  
 $\text{exd} \equiv 1 \pmod{\phi(n)} \Rightarrow d = (k \times \phi(n) + 1) / e$

The keys are:

**K<sub>p</sub> = {e, n} (public) and**

**K<sub>u</sub> = {d, n} (private)**

### A Numeric Example

Following the steps of the algorithm:

1. Choose 2 prime numbers **p** and **q** in accordance with the algorithm characteristics. Let

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

$$q = 1019 = 1111111011_2$$

2. Calculate **n**, such that

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

$$1040399 = 1111111000000001111_2$$

3. Calculate **φ(n)** such that

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

$$1038360 = 11111101100000011000_2$$

4. Choose **e = 3577 = 110111111001<sub>2</sub>** satisfying

$$\text{mdc}(3577, 1038360) = 1$$

5. Finally, calculate **d**, satisfying

$$1 < d < \phi(n) \text{ and } \text{exd} \equiv 1 \pmod{\phi(n)}$$

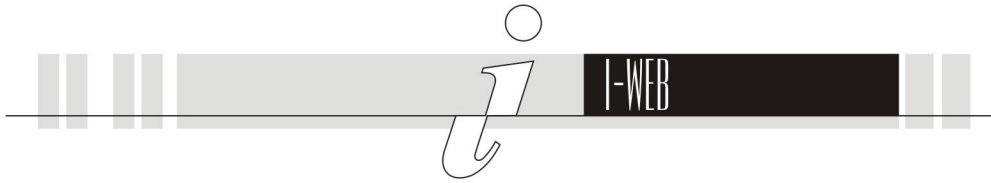
$$d = 426433 = 01101000000111000001_2$$

The cryptographic keys are:

**K<sub>p</sub> = {3577, 1040399} (public key)**

**K<sub>u</sub> = {426433, 1040399} (private key)**

As **n** has 20 bits, we say that this is an algorithm with a 20 bits long key or a 20 bits key.



## RSA: PUBLIC-KEY ALGORITHMS

FIRST PUBLISHING: APRIL/1998

FOURTH REVISION: DECEMBER/2004

The message has to be transmitted in blocks of length  $n$ , i.e.,  $0 < M < n$ . Thus, in this example, we can only transmit messages up to 20 bits or we have to divide the message  $M$  in blocks  $M_1, M_2, \dots, M_N$  (such that  $M = M_1|M_2|\dots|M_N$ ), where  $M_i$  is 20 bits long.

### Modular Arithmetic

$b \mid a$

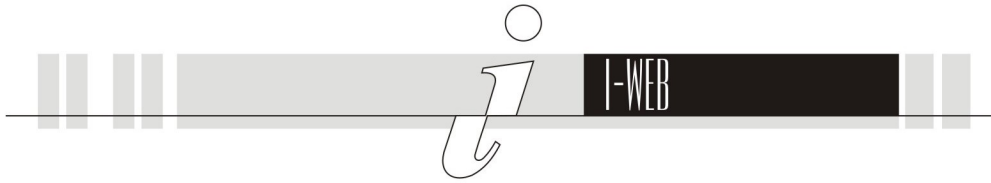
**Definition:** If  $a$  and  $b$  are non-negative integers, then  $b$  is said to “divide”  $a$ , written  $b \mid a$ , if exists one integer  $k$  satisfying the equation  $a = k \times b$ .

$a \equiv r \pmod{b}$

Using the above definition, each integer can be written as  $a = k \times b + r$ , where:  $b, k$  and  $r$  are integers and  $0 \leq r < b$  and we say that  $a \equiv r \pmod{b}$ , i.e., there is one integer  $k$  satisfying  $(a-r) = k \times b$ . The integer  $b$  is called modulus of congruence.

**Property 1:**

The above equation gives us a very interesting property:  
 $a \equiv r \pmod{b} \Rightarrow (a-r) = k \times b \Rightarrow -1 \times (a-r) = -1 \times k \times b \Rightarrow (r-a) = (-1 \times k) \times b \Rightarrow r \equiv a \pmod{b}$ .



## RSA: PUBLIC-KEY ALGORITHMS

FIRST PUBLISHING: APRIL/1998

FOURTH REVISION: DECEMBER/2004

### Euler Phi (or Totient) Function

#### Euler Phi Function or (Euler Totient Function)

Let us consider the function

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

or putting in words, “the number of integers in the interval  $[1, n]$  which are relatively prime to  $n$ ”.

#### Property 2:

Let  $p$  be a prime number.  $\phi(p) = p-1$ . This follows by the definition of prime numbers, as for each  $p$  prime,  $\forall k \in \mathbb{Z}_+, 0 < k < p$ ,  $k$  does not divide  $p$  (otherwise,  $p$  would be a prime number).

#### Property 3:

Let  $p$  and  $q$  be 2 prime numbers. Let  $n = p \times q$ .  
We will calculate  $\phi(n)$ .

Taking  $n = p \times q$ , the divisors of  $n$  are numbers in the form:

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

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

$$\text{Therefore, } |P| = (q-1) \text{ and } |Q| = (p-1).$$

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

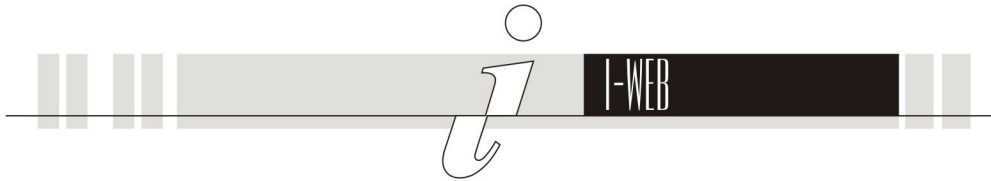
#### Property 4:

Now, we will consider the Euler Theorem:  $m^{\phi(n)} \equiv 1 \pmod{n}$ .

It means that exists one integer  $k$  satisfying  $(m^{\phi(n)} - 1) = k \times n$ .

Alternatively, we can write that  $m \times (m^{\phi(n)} - 1) = m \times (k \times n) \Rightarrow$

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



## RSA: PUBLIC-KEY ALGORITHMS

FIRST PUBLISHING: APRIL/1998

FOURTH REVISION: DECEMBER/2004

### Implementation

#### Algorithms

The algorithms used in the process to obtain the RSA elements are quite simple, however, considering we are manipulating large numbers, it is necessary to use appropriate algorithms to manipulate them with the necessary accuracy.

*Obs: The following algorithms are presented without any optimization and for didactical purposes only, in order to help the understanding.*

#### Calculating the primes p and q

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

$p$  is prime  $\Leftrightarrow$  there is no such  $k \in \mathbb{Z}_+$  such that  $1 < k < p$  where  $k \mid p$ .

Note that 1 is not a prime number (in agreement with the definition).

We can be a little more restrictive in the definition of a prime number, because if  $k, x \in \mathbb{Z}_+$  e  $k \mid x$ , then  $x = k \times j$  and therefore  $j \mid x$ .

Let  $n = \min(k, j)$  and  $m = \max(k, j)$ .

$0 < n \leq \sqrt{x}$ , otherwise, we would have  $m \geq n$ ,  $m \times n > \sqrt{x} \times \sqrt{x} > x$ .

Then  $p$  is prime  $\Leftrightarrow$  there is no such  $k \in \mathbb{Z}_+$  with  $1 < k \leq \sqrt{p}$  such that  $k \mid p$ .

Let  $\mathbb{Z}_p = \{c \in \mathbb{Z}_+ \mid 1 < c < p \text{ and } c \text{ is a prime number}\}$

Analyzing the property of division, we can note that:

Let  $c \in \mathbb{Z}_p$  and  $k, x \in \mathbb{Z}_+$ ,

if  $c \mid k$  and  $k \mid x \Rightarrow k = i \times c$  and  $x = k \times j \Rightarrow x = (i \times c) \times j = (i \times j) \times c \Rightarrow c \mid x$  where  $i, j \in \mathbb{Z}_+$ .

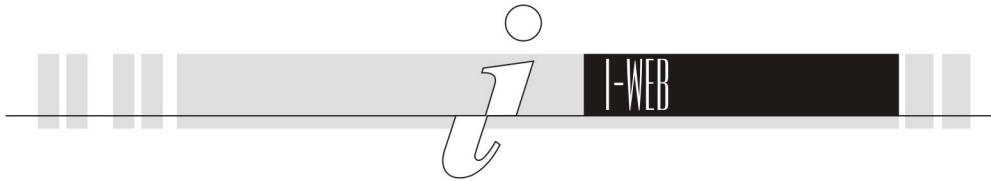
Hence, we can define  $p$  prime as:

$p$  is a prime number  $\Leftrightarrow$  there is no such  $k \in \mathbb{Z}_p$  with  $1 < k \leq \sqrt{p}$  such that  $k \mid p$ .

Meaning that we just have to try for all the primes lesser than  $p$ .

The algorithm below identify if a number  $p$  is a prime number, trying to divide it by each integer  $k$ , such that  $1 < k \leq \sqrt{p}$ .

```
#include <math.h>
int pprimo(p)
{
    int k,sqrtp;
    sqrtp = sqrt(p);
    k=2;
```



## RSA: PUBLIC-KEY ALGORITHMS

FIRST PUBLISHING: APRIL/1998

FOURTH REVISION: DECEMBER/2004

```
while ((k<=sqrtp) && ((p%k) != 0))
k++;
if (k >= sqrtp) return 1;
else return 0;
}
```

Obs: Although the algorithm above try for “all”  $k \in \mathbb{Z}$  such that  $1 < k \leq \sqrt{p}$ , in fact we just need to try for each prime  $k$  ( $k \in \mathbb{Z}_p$ ). We choose this because it makes simple the algorithm implementation, but this is very inefficient when using huge numbers. In a real case, we could keep a list with at least the first 2000 prime numbers, for example. However, decide if a number  $n$  is prime is can be a very complex task (which is, in fact, the goal of this algorithm).

### Choosing e

Only 2 properties must be satisfied during the choice of e:

- i.  $1 < e < \phi(n)$  and
- ii.  $\gcd(e, \phi(n)) = 1$ .

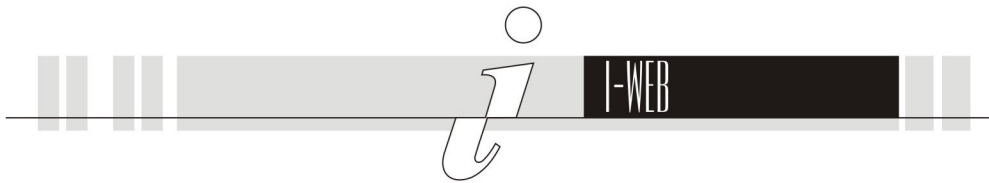
The easiest way to calculate **e**, therefore, is choose **e** in the prime factorization of  $\phi(n)$ . In other words, we can choose **e** among the primes with  $\phi(n)$ .

Prime factorization of  $\phi(n)$  can be a not so easy task, as a result this method may not be very efficient (may be poorly efficient) one. To make the task easier, we can choose e among the prime numbers lesser than  $\phi(n)$ . This choice gives an additional benefit (and simplify the test), because if p is a prime number,  $\gcd(p, \phi(n)) \in \{1, p\}$ .

We also can note that if p and q  $\neq 2$ , so that p and q are odd number and, therefore,  $(p-1) \times (q-1)$  is even. Thus  $e \neq 2 \times k$  for any integer k.

We describe this particular case implementation below.

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



## RSA: PUBLIC-KEY ALGORITHMS

FIRST PUBLISHING: APRIL/1998

FOURTH REVISION: DECEMBER/2004

### Calculating d

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

$d = (k \times \phi(n) + 1) / e$ , i.e., d is the multiplicative inverse of e mod n.

Considering how n has been chosen, given e, there only one d satisfying the equation.

Therefore, we choose a “candidate” for d and then calculate  $r = (d \times e) \pmod{\phi(n)}$ . If  $r = 1$ , d is valid.

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

### Bit conversion

One of the first algorithms one can learn is the conversion number representation. In this case from base-10 to binary.

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

### Calculating $m^e \pmod n$

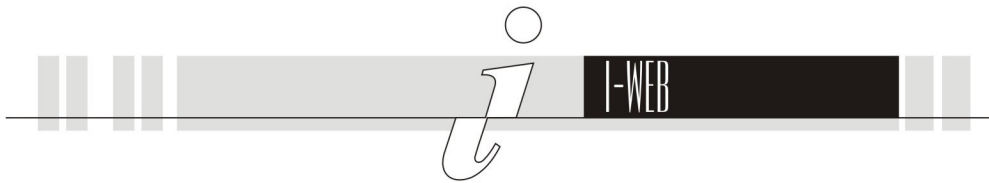
The calculation process of  $C = M^e \pmod n$  is based on the exponentiation of M by e. There are several ways to calculate  $M^e$ , such as:

$M^{16} = M \times M \times \dots \times M$  corresponding to 15 multiplications by M or using a more economic approach:

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

which brings to the same result performing only 4 multiplications.

Calculating  $M^e$  easily can go beyond the computer capacity for numeric manipulation. Thus, it is necessary to proceed carefully when calculating  $M^e$ .



## RSA: PUBLIC-KEY ALGORITHMS

FIRST PUBLISHING: APRIL/1998

FOURTH REVISION: DECEMBER/2004

```
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;
}
```

Despite the procedure above, the computer can not be able to manipulate such big numbers. For the example presented in this paper, it can be necessary to use “long double” types instead of “int” types.

The latter algorithm becomes:

```
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: When calculating 128-bit or greater keys even the “long double” type can be insufficient for the appropriate numeric manipulation. In these cases, it is necessary to use other numeric implementations in order to manipulate those numbers accordantly.